

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

PCI-8132

2 Axes Servo / Stepper Motion Control Card

User's Guide



Recycled Paper

@Copyright 2006 ADLINK Technology Inc.
All Rights Reserved.

Manual Rev. 2.01: Jan 17, 2006

The information in this document is subject to change without prior notice in order to improve reliability, design and function and does not represent a commitment on the part of the manufacturer.

In no event will the manufacturer be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the product or documentation, even if advised of the possibility of such damages.

This document contains proprietary information protected by copyright. All rights are reserved. No part of this manual may be reproduced by any mechanical, electronic, or other means in any form without prior written permission of the manufacturer.

Trademarks

NuDAQ, PCI-8132 are registered trademarks of ADLINK Technology Inc, MS-DOS , Windows 95/98 , Windows NT/2000 are registered trademarks of Microsoft Corporation., Borland C++ is a registered trademark of Borland International, Inc. Other product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

Getting service from ADLINK

♦Customer Satisfaction is always the most important thing for ADLINK Tech Inc. If you need any help or service, please contact us and get it.

ADLINK Technology Inc.			
Web Site	http://www.adlinktech.com		
Sales & Service	service@adlinktech.com		
Technical Support	NuDAQ	nudaq@adlinktech.com	
	Automation	automation@adlinktech.com	
	NuIPC	nuipc@adlinktech.com	
	NuPRO/EBC	nupro@adlinktech.com	
TEL	+886-2-82265877	FAX	+886-2-82265717
Address	9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan		

♦Please inform or FAX us of your detailed information for a prompt, satisfactory and constant service.

Detailed Company Information			
Company/Organization			
Contact Person			
E-mail Address			
Address			
Country			
TEL		FAX	
Web Site			

Questions	
Product Model	
Environment to Use	☐OS _____ ☐Computer Brand _____ ☐M/B: ☐CPU: ☐Chipset: ☐Bios: ☐Video Card: ☐Network Interface Card: ☐Other:
Challenge Description	
Suggestions for ADLINK	

Table of Contents

Chapter 1 Introduction 1

1.1 Features 4
1.2 Specifications 5
1.3 Software Supporting 6

Chapter 2 Installation 7

2.1 What You Have 7
2.2 PCI-8132 Outline Drawing 8
2.3 Hardware Installation 9
2.3.1 Hardware configuration 9
2.3.2 PCI slot selection 9
2.3.3 Installation Procedures 9
2.3.4 Trouble shooting: 9
2.4 Software Driver Installation 10
2.5 CN1 Pin Assignments: External Power Input 10
2.6 CN2 Pin Assignments: Main connector 11
2.7 CN3 Pin Assignments: Simultaneous Start/Stop 12
2.8 Jumper Setting 13
2.9 Switch Setting 13

Chapter 3 Signal Connections 14

3.1 Pulse Output Signals OUT and DIR 15
3.2 Encoder Feedback Signals EA, EB and EZ 17
3.3 Origin Signal ORG 19
3.4 End-Limit Signals PEL and MEL 20
3.5 Ramping-down Signals PSD and MSD 21
3.6 In-position Signal INP 22
3.7 Alarm Signal ALM 23
3.8 Deviation Counter Clear Signal ERC 24
3.9 General-purpose Signal SVON 25
3.10 General-purpose Signal RDY 26
3.11 Isolated Digital Output DOx 27
3.12 Isolated Digital Input DIx 28
3.13 Pulser Input Signals PA and PB 29
3.14 Simultaneously Start/Stop Signals STA and STP 30
3.15 Daughter Board Connector 31
3.16 Comparison Output CMP1 and CMP2 32

Chapter 4 Operation Theorem 33

4.1 Motion Control Modes 33
4.1.1 Pulse Command Output 34
4.1.2 Constant Velocity Motion 35
4.1.3 Trapezoidal Motion 36
4.1.4 S-curve Profile Motion 40
4.1.5 Linear and Circular Interpolated Motion 43
4.1.6 Home Return Mode 44
4.1.7 Manual Pulser Mode 46
4.2 Motor Driver Interface 47
4.2.1 INP 47
4.2.2 ALM 47
4.2.3 ERC 48
4.3 The Limit Switch Interface and I/O Status 49
4.3.1 SD 49
4.3.2 EL 49
4.3.3 ORG 50
4.3.4 SVON and RDY 50
4.4 The Encoder Feedback Signals (EA, EB, EZ) 50
4.5 Multiple PCI-8132 Cards Operation 52
4.6 Change Speed on the Fly 53
4.7 Position Comparison 55
4.8 Interrupt Control 59

Chapter 5 Motion Creator 63

5.1 Main Menu 64
5.2 Axis Configuration Window 65
5.3 Axis Operation Windows 68
5.3.1 Motion Status Display 68
5.3.2 Axis Status Display 68
5.3.3 I/O Status Display 69
5.3.4 Set Position Control 69
5.3.5 Operation Mode Control 69
5.3.6 Motion Parameters Control 70
5.3.7 Play Key Control 70
5.3.8 Velocity Profile Selection 71
5.3.9 Repeat Mode 71
5.4 2-D Motion Windows ..... 錯誤! 尚未定義書籤。
5.4.1 Linear Interpolation ..... 錯誤! 尚未定義書籤。
5.4.2 Circular Interpolation ..... 錯誤! 尚未定義書籤。
5.4.3 Continuous Jog ..... 錯誤! 尚未定義書籤。
5.4.4 Incremental Jog ..... 錯誤! 尚未定義書籤。

5.4.5 Other Control Objects .....錯誤! 尚未定義書籤。

Chapter 6 Function Library.....73

6.1 List of Functions.....73
6.2 C/C++ Programming Library.....77
6.3 Initialization.....78
6.4 Pulse Input / Output Configuration.....81
6.5 Continuously Motion Move.....82
6.6 Trapezoidal Motion Mode.....83
6.7 S-Curve Profile Motion.....86
6.8 Multiple Axes Point to Point Motion 88
6.9 Linear and Circular Interpolated Motion 90
6.10 Interpolation Parameters Configuring.....90
6.11 Home Return 92
6.12 Manual Pulser Motion.....93
6.13 Motion Status 95
6.14 Servo Drive Interface.....97
6.15 I/O Control and Monitoring 99
6.16 Position Control.....100
6.17 Interrupt Control 101
6.18 Digital Input/Output Control 104
6.19 Position Compare Control 105

Chapter 7 Connection Example 108

7.1 General Description of Wiring.....108
7.2 Connection Example with Servo Driver 111

Product Warranty/Service..... 錯誤! 尚未定義書籤。

How to Use This Guide

This manual is designed to help you use the PCI-8132. The manual describes how to modify various settings on the PCI-8132 card to meet your requirements. It is divided into seven chapters:

- **Chapter 1**, "Introduction", gives an overview of the product features, applications, and specifications.
- **Chapter 2**, "Installation", describes how to install the PCI-8132.
- **Chapter 3**, "Signal Connection", describes the connectors' pin assignment and how to connect the outside signal and devices with the PCI-8132.
- **Chapter 4**, "Operation Theorem", describes detail operations of the PCI-8132.
- **Chapter 5**, “Motion Creator”, describe how to utilize a Microsoft Windows based utility program to configure and test running the PCI-8132
- **Chapter 6**, " Function Library", describes high-level programming interface in C/C++ and VB language. It helps programmer to control PCI-8132 in high level language style.
- **Chapter 7,** “Connection Example” shows some typical connection examples between PCI-8132 and servo driver and stepping driver.

1

Introduction

The PCI-8132 is a 2 axes motion control card with PCI interface. It can generate high frequency pulses to drive stepping motors and servo motors. Multiple PCI-8132 cards can be used in one system. Incremental encoder interface on all four axes provide the ability to correct for positioning errors generated by inaccurate mechanical transmissions. In addition, mechanical sensor interface, servo motor interface and general purpose I/O signals are provided for system integration. Hardware position compare function and trigger signal output provide users a way of taking pictures while the motors are still in motion.

Figure 1.1 shows the function block diagram of PCI-8132 card. PCI-8132 uses one ASIC (PCL5023) to perform 2 axes motion control. This ASIC is made of Nippon Pulse Motor incooperation. The motion control functions include linear and S-curve acceleration/deceleration, interpolation between two axes, continuous motion, in positioning and home return are done by the ASIC. Since these functions needing complex computations are done internally on the ASIC, the PC's CPU is free to supervise and perform other tasks.

Motion Creator, a Microsoft Windows based software is equipped with the PCI-8132 card for supporting application development. The Motion Creator is very helpful for debugging a motion control system during the design phase of a project. The on-screen monitor shows all installed axis information and I/O signals status of PCI-8132 cards. In addition to Motion Creator, both DOS and Windows version function library are included for programmers using C++ and Visual Basic language. Several sample programs are given to illustrate how to use the function library.

Figure 1.2 is a flowchart that shows a recommending process of using this manual to develop an application. Please also refer the relative chapters for the detail of each step.

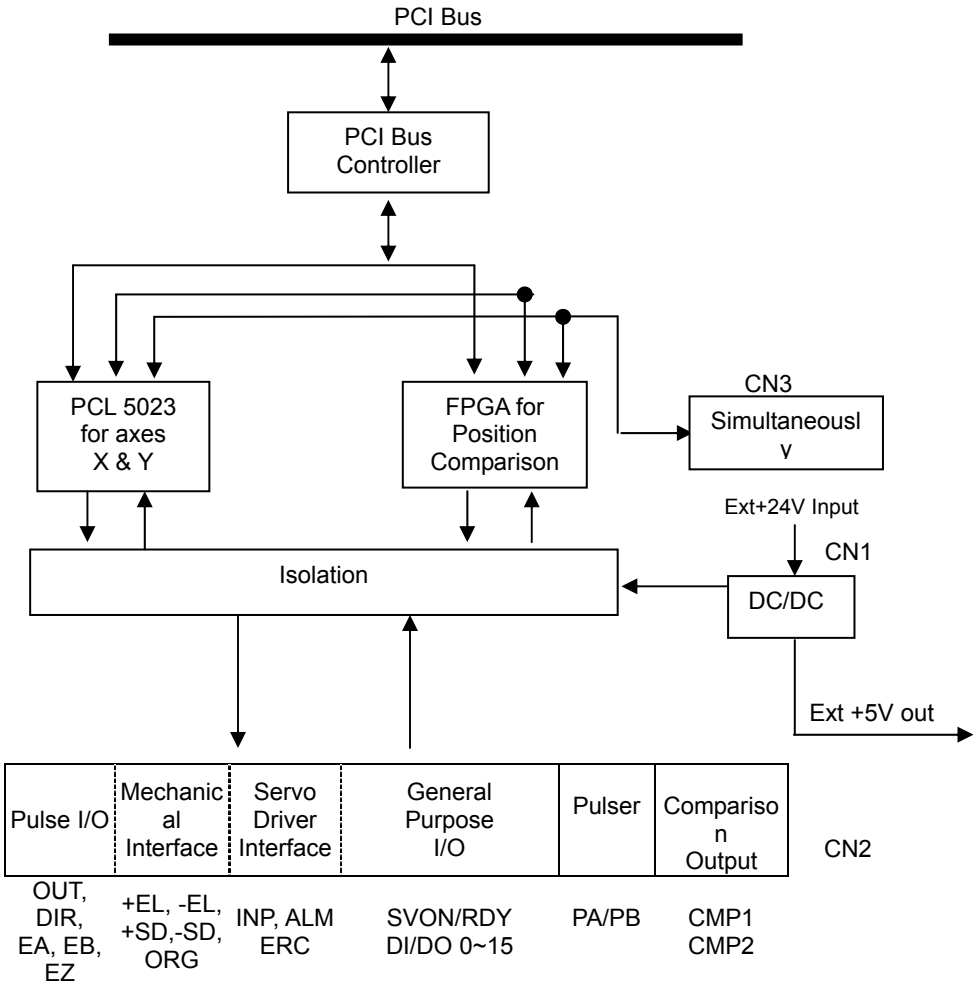


Figure 1.1 Block Diagram of PCI-8132

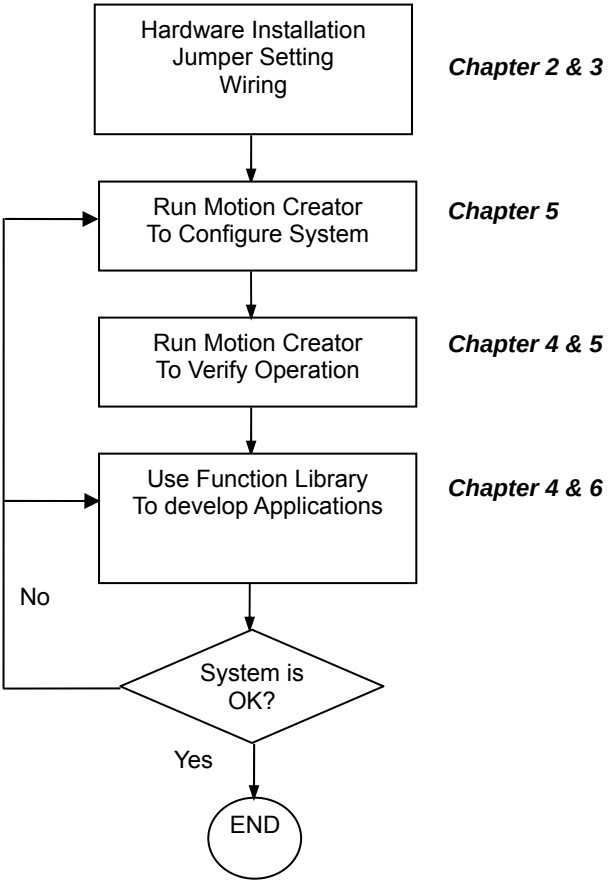


Figure 1.2 Flowchart of building an application

1.1 Features

The following lists summarize the main features of the PCI-8132 motion control system.

- 32-bit PCI-Bus, plug and play.
- 2 axes of step and direction pulse output for controlling stepping or servomotor.
- Maximum output frequency of 2.4 Mpps.
- 28-bit up/down counter for incremental encoder feedback.
- Home switch, index signal, positive and negative limit switches interface provided for all axes.
- Programmable interrupt sources.
- Change Speed on the Fly.
- Positiion Compare and Trigger Signal output.
- Simultaneous start/stop motion on multiple axes.
- Manual pulser input interface.
- Software supports maximum up to 12 PCI-8132 cards (24 axes) operation.
- Compact, half size PCB.
- Motion Creator, Microsoft Windows based application development software.
- PCI-8132 Library and Utility for DOS library and Windows 95/98/NT/2000/XP DLL.

1.2 Specifications

- **Applicable Motors:**
  - Stepping motors.
  - AC or DC servomotors with pulse train input servodrivers.
- **Performance:**
  - Number of controllable axes: 2 axes.
  - Maximum pulse output frequency: 2.4Mpps, linear, trapezoidal or S-Curve velocity profile drive.
  - Internal reference clock: 9.8304 MHz
  - Position pulse setting range: 0~268,435,455 pulses(28-bit).
  - Up / down counter counting range: 0~268,435,455 (28-bit.) or –134,217,728 to +134,217,727
  - Pulse rate setting steps: 0 to 2.4Mpps.
  - Position Comparison Range:-8,388,608 ~ +8388607(24 bits)
- **I/O Signales:**
  - Input/Output Signals for each axis
  - All I/O signal are optically isolated with 2500Vrms isolation voltage
  - Command pulse output pins: OUT and DIR.
  - Incremental encoder signals input pins: EA and EB.
  - Encoder index signal input pin: EZ.
  - Mechanical limit/switch signal input pins: ±EL, ±SD and ORG.
  - Servomotor interface I/O pins: INP, ALM and ERC.
  - General purpose digital output pin: SVON.
  - General purpose digital input pin: RDY.
  - Pulser signal input pin: PA and PB.
  - Simultaneous Start/Stop signal I/O pins: STA and STP.
  - 16 Channels Open collector digital output
  - 16 Channels Isolated digital input
  - Trigger Output Signals:CMP1/CMP2
  -
- **General Specifications**
  - Connectors: 100-pin SCSI-type connector
  - Operating Temperature: 0° C ~ 50° C
  - Storage Temperature: -20° C ~ 80° C
  - Humidity: 5 ~ 85%, non-condensing
  - Power Consumption:
    - ✧ Slot power supply(input): +5V DC ±5%, 900mA max.

- ✧ External power supply(input): +24V DC ±5%, 500mA max.
- ✧ External power supply(output): +5V DC ±5%, 500mA, max.

- Dimension: 164mm(L) X 98.4mm(H)

1.3 Software Supporting

For the customers who are writing their own programs, we provide MS-DOS Borland C/C++ programming library and Windows 95/98/NT/2000/XP DLL for PCI-8132. These function libraries are shipped with the board.

2

Installation

This chapter describes how to install the PCI-8132. Please follow the follow steps to install the PCI-8132.

- Check what you have (section 2.1)
- Check the PCB (section 2.2)
- Install the hardware (section 2.3)
- Install the software driver (section 2.4)
- Understanding the I/O signal connections (chapter 3) and their operation (chapter 4)
- Understanding the connectors' pin assignments (the rest of the sections) and wiring the connections

2.1 What You Have

In addition to this *User's Guide*, the package includes the following items:

- PCI-8132 2 axes Servo / Stepper Motion Control Card
- ADLINK CD
- User's Manual
- +24V power input cable (for CN1)

If any of these items are missing or damaged, contact the dealer from whom you purchased the product. Save the shipping materials and carton in case you want to ship or store the product in the future.

2.2 PCI-8132 Outline Drawing

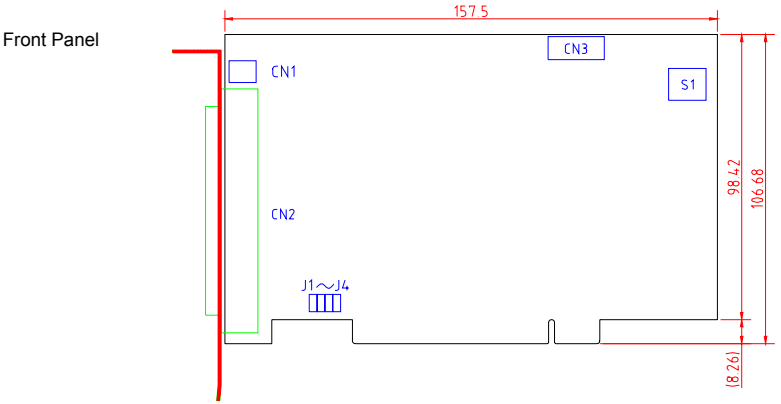
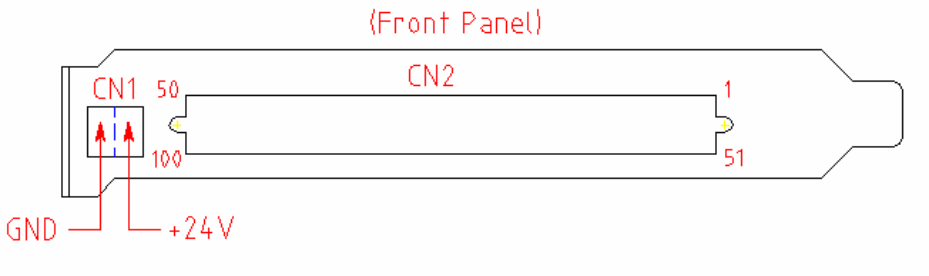


Figure 2.1 PCB Layout of the PCI-8132

CN1: External Power Input Connector
CN2: Input / Output Signal Connector
CN3: Simultaneous Start/Stop



2.3 Hardware Installation

2.3.1 Hardware configuration

PCI-8132 has plug and play PCI controller on board. The memory usage (I/O port locations) of the PCI card is assigned by system BIOS. The address assignment is done on a board-by-board basis for all PCI cards in the system.

2.3.2 PCI slot selection

Your computer will probably have both PCI and ISA slots. Do not force the PCI card into a PC/AT slot. The PCI-8132 can be used in any PCI slot.

2.3.3 Installation Procedures

1. Read through this manual, and setup the jumper according to your application
2. Turn off your computer, Turn off all accessories (printer, modem, monitor, etc.) connected to computer.
   Remove the cover from your computer.
3. Select a 32-bit PCI expansion slot. PCI slots are short than ISA or EISA slots and are usually white or ivory.
4. Before handling the PCI-8132, discharge any static buildup on your body by touching the metal case of the computer. Hold the edge and do not touch the components.
5. Position the board into the PCI slot you selected.
6. Secure the card in place at the rear panel of the system unit using screw removed from the slot.

2.3.4 Trouble shooting:

If your system won‘t boot or if you experience erratic operation with your PCI board in place, it’s likely caused by an interrupt conflict (perhaps because you incorrectly described the ISA setup). In general, the solution, once you determine it is not a simple oversight, is to consult the BIOS documentation that come with your system.

2.4 Software Driver Installation

Please refer to the PCI Software Installation Guide.

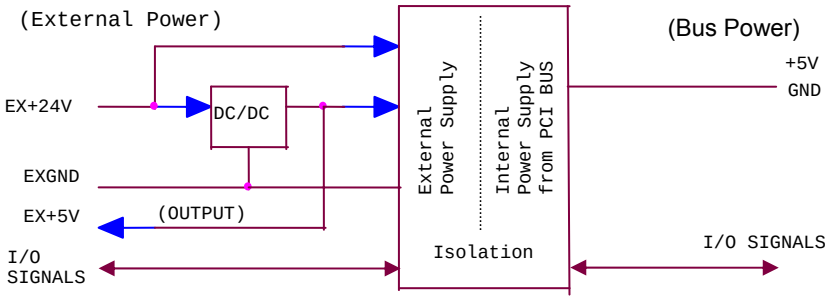
2.5 CN1 Pin Assignments: External Power Input

CN1 Pin No	Name	Description
1	EXGND	Grounds of the external power.
2	EX+24V	External power supply of +24V DC ± 5%

Notes:

1. CN1 is a plug-in terminal board with no screw.
2. Be sure to use the external power supply. The +24V DC is used by external input/output signal circuit. The power circuit is configured as follows.
3. Wires for connection to CN1
   Solid wire: φ 0.32mm to φ 0.65mm (AWG28 to AWG22)
   Twisted wire:0.08mm^2 to 0.32mm^2 (AWG28 to AWG22)
   Naked wire length:10mm standard

The following diagram shows the external power supply system of the PCI-8132. The external +24V power must be provided, an on-board regulator generates +5V for both internal and external usage.



2.6 CN2 Pin Assignments: Main connector

The CN2 is the major connector for the motion control I/O signals.

No.	Name	I/O	Function(axis①/②)	No.	Name	I/O	Function(axis③/④)
1	VPP +5V	O	+5V power supply output	51	DO COM+	I	Ext power input for Dout
2	EXGND	O	Ext. power ground	52	EXGND	O	Ext. power ground
3	OUT1+	O	Pulse signal (+),①	53	DO0	O	Isolated digital output 0
4	OUT1-	O	Pulse signal (-),①	54	DO1	O	Isolated digital output 1
5	DIR1+	O	Dir. signal (+),①	55	DO2	O	Isolated digital output 2
6	DIR1-	O	Dir. signal (-),①	56	DO3	O	Isolated digital output 3
7	SVON1	O	Multi-purpose signal, ①	57	DO4	O	Isolated digital output 4
8	ERC1	O	Dev. ctr, clr. signal, ①	58	DO5	O	Isolated digital output 5
9	ALM1	I	Alarm signal, ①	59	DO6	O	Isolated digital output 6
10	INP1	I	In-position signal, ①	60	DO7	O	Isolated digital output 7
11	RDY1	I	Multi-purpose signal, ①	61	DO8	O	Isolated digital output 8
12	EXGND	O	Ext. power ground	62	DO9	O	Isolated digital output 9
13	EA1+	I	Encoder A-phase (+), ①	63	DO10	O	Isolated digital output 10
14	EA1-	I	Encoder A-phase (-), ①	64	DO11	O	Isolated digital output 11
15	EB1+	I	Encoder B-phase (+), ①	65	DO12	O	Isolated digital output 12
16	EB1-	I	Encoder B-phase (-), ①	66	DO13	O	Isolated digital output 13
17	EZ1+	I	Encoder Z-phase (+), ①	67	DO14	O	Isolated digital output 14
18	EZ1-	I	Encoder Z-phase (-), ①	68	DO15	O	Isolated digital output 15
19	VPP +5V	O	+5V power supply output	69	EXGND	O	Ext. power ground
20	EXGND	O	Ext. power ground	70	EXGND	O	Ext. power ground
21	OUT2+	O	Pulse signal (+), ②	71	DI COM+	I	Ext power input for Din
22	OUT2-	O	Pulse signal (-), ②	72	DI COM+	I	Ext power input for Din
23	DIR2+	O	Dir. signal (+), ②	73	DI0	I	Isolated digital input 0
24	DIR2-	O	Dir. signal (-), ②	74	DI1	I	Isolated digital input 1
25	SVON2	O	Multi-purpose signal, ②	75	DI2	I	Isolated digital input 2
26	ERC2	O	Dev. ctr, clr. signal, ②	76	DI3	I	Isolated digital input 3
27	ALM2	I	Alarm signal, ②	77	DI4	I	Isolated digital input 4
28	INP2	I	In-position signal, ②	78	DI5	I	Isolated digital input 5
29	RDY2	I	Multi-purpose signal, ②	79	DI6	I	Isolated digital input 6
30	EXGND	O	Ext. power ground	80	DI7	I	Isolated digital input 7
31	EA2+	I	Encoder A-phase (+), ②	81	DI8	I	Isolated digital input 8
32	EA2-	I	Encoder A-phase (-), ②	82	DI9	I	Isolated digital input 9
33	EB2+	I	Encoder B-phase (+), ②	83	DI10	I	Isolated digital input 10
34	EB2-	I	Encoder B-phase (-), ②	84	DI11	I	Isolated digital input 11
35	EZ2+	I	Encoder Z-phase (+), ②	85	DI12	I	Isolated digital input 12
36	EZ2-	I	Encoder Z-phase (-), ②	86	DI13	I	Isolated digital input 13
37	PEL1	I	End limit signal (+), ①	87	DI14	I	Isolated digital input 14
38	MEL1	I	End limit signal (-), ①	88	DI15	I	Isolated digital input 15
39	PSD1	I	Ramp-down signal (+), ①	89	EXGND	I	Ext. power ground
40	MSD1	I	Ramp-down signal (-), ①	90	EXGND	I	Ext. power ground
41	ORG1	I	Origin signal, ①	91	PA+	I	Manual Pulser Input PHA+
42	EXGND	O	Ext. power ground	92	PA-	I	Manual Pulser Input PHA-
43	PEL2	I	End limit signal (+), ②	93	PB+	I	Manual Pulser Input PHB+
44	MEL2	I	End limit signal (-), ②	94	PB-	I	Manual Pulser Input PHB-
45	PSD2	I	Ramp-down signal (+), ②	95	EXGND	I	Ext. power ground
46	MSD2	I	Ramp-down signal (-), ②	96	CMP1	O	Position compare Trigger 1
47	ORG2	I	Origin signal, ②	97	CMP2	O	Position compare Trigger 2
48	EXGND	O	Ext. power ground	98	EXGND	O	Ext. power ground
49	EXGND	O	Ext. power ground	99	VPP +24V	O	+24V power supply output
50	EXGND	O	Ext. power ground	100	VPP +24V	O	+24V power supply output

2.7 CN3 Pin Assignments: Simultaneous Start/Stop

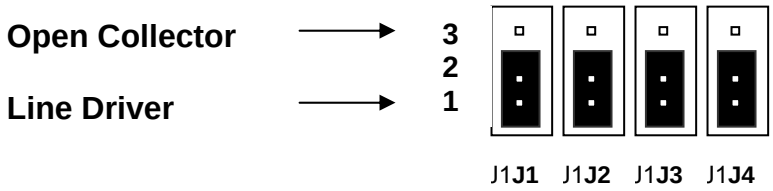
The signals on CN3 is for simultaneously start/stop signals for multiple axes and multiple cards.

No.	Name	Function(Axis)
1	GND	Bus power ground
2	STP	Simultaneous stop signal input/output
3	STA	Simultaneous start signal input/output
4	STP	Simultaneous stop signal input/output
5	STA	Simultaneous start signal input/output
6	+5V	Bus power, +5V

Note: +5V and GND pins are directly given by the PCI Bus power.

2.8 Jumper Setting

The J1~J4 is used to set the signal type of the pulse output signals (DIR and OUT). The output signal type could be differential line driver output or open collector output. Please refer to section 3.1 for details of the jumper setting. The default setting is the differential line driver mode.

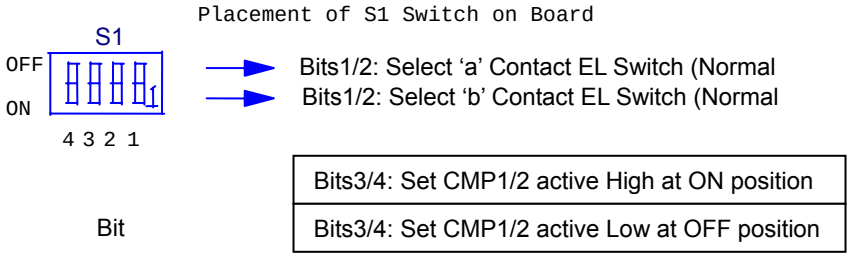


2.9 Switch Setting

The switch bits 1/2 of S1 are used to set the EL limit switch's type. The default setting of EL switch type is "normal open" type limit switch (or "A" contact type). The switch on is to use the "normal closed" type limit switch (or "B" contact type). The default setting is set as normal open type.

The bits 3/4 of S1 are used to set the active logic of CMP1,CMP2 respectively.

Default setting is active low. This means that when a positive comparison condition is met ,CMP will go high for 100 us automatically.



3

Signal Connections

The signal connections of all the I/O signals are described in this chapter. Please refer the contents of this chapter before wiring the cable between the PCI-8132 and the motor drivers.

This chapter contains the following sections:

Section 3.1	Pulse output signals OUT and DIR
Section 3.2	Encoder feedback signals EA, EB and EZ
Section 3.3	Origin signal ORG
Section 3.4	End-Limit signals PEL and MEL
Section 3.5	Ramping-down signals PSD and MSD
Section 3.6	In-position signal INP
Section 3.7	Alarm signal ALM
Section 3.8	Deviation counter clear signal ERC
Section 3.9	General-purpose signal SVON
Section 3.10	General-purpose signal RDY
Section 3.11	General Purpose Digital Output
Section 3.12	General Purpose Digital Input
Section 3.13	Pulser input signals PA and PB
Section 3.14	Simultaneous start/stop signals STA and STP
Section 3.15	Comparison Output CMP1,CMP2
Section 3.16	Daughter Board Connector

3.1 Pulse Output Signals OUT and DIR

There are 2-axis pulse output signals on PCI-8132. For every axis, two pairs of OUT and DIR signals are used to send the pulse train and to indicate the direction. The OUT and DIR signals can also be programmed as CW and CCW signals pair, refer to section 4.1.1 for details of the logical characteristics of the OUT and DIR signals. In this section, the electronic characteristics of the OUT and DIR signals are shown. Each signal consists of a pair of differential signals. For example, the OUT2 is consisted of OUT2+ and OUT2- signals. The following table shows all the pulse output signals on CN2.

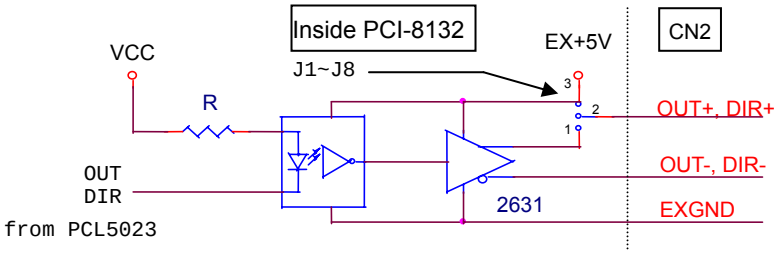
CN2 Pin No.	Signal Name	Description	Axis #
3	**OUT1+**	Pulse signals (+)	①
4	**OUT1-**	Pulse signals (-)	①
5	**DIR1+**	Direction signal(+)	①
6	**DIR1-**	Direction signal(-)	①
21	**OUT2+**	Pulse signals (+)	②
22	**OUT2-**	Pulse signals (-)	②
23	**DIR2+**	Direction signal(+)	②
24	**DIR2-**	Direction signal(-)	②

The output of the OUT or DIR signals can be configured by jumpers as either the differential line driver or open collector output. You can select the output mode either by closing breaks between 1 and 2 or 2 and 3 of jumpers J1~J4 as follows.

Output Signal	For differential line driver output, close a break between 1 and 2 of	For open collector output, close a break between 2 and 3 of:
OUT1-	J1	J1
DIR1-	J2	J2
OUT2-	J3	J3
DIR2-	J4	J4

The **default** setting of OUT and DIR signals are the as differential line driver mode.

The following wiring diagram is for the OUT and DIR signals of the 2 axes.



NOTE: If the pulse output is set to the open collector output mode, the OUT- and DIR- are used to send out signals. Please take care that the current sink to OUT- and DIR- pins must not exceed 20mA. The current may provide by the EX+5V power source, however, please note that the maximum capacity of EX+5V power is 500mA.

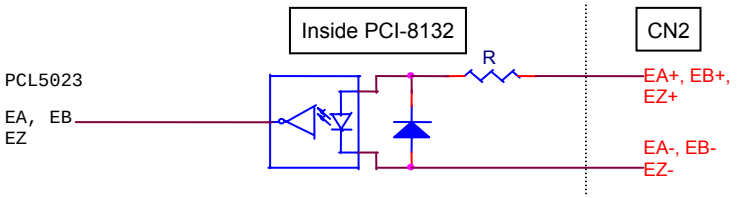
3.2 Encoder Feedback Signals EA, EB and EZ

The encoder feedback signals include the EA, EB, and EZ. Every axis has six pins for three differential pairs of phase-A (EA), phase-B (EB) and index (EZ) input. The EA and EB are used for position counting, the EZ is used for zero position index. The relative signal names, pin numbers and the axis number are shown in the following tables.

CN2 Pin No	Signal Name	Axis #	CN2 Pin No	Signal Name	Axis #
13	EA1+	①	31	EA2+	②
14	EA1-	①	32	EA2-	②
15	EB1+	①	33	EB2+	②
16	EB1-	①	34	EB2-	②

CN2 Pin No	Signal Name	Axis #	CN2 Pin No	Signal Name	Axis #
17	EZ1+	①	35	EZ2+	②
18	EZ1-	①	36	EZ2-	②

The input circuits of the EA, EB, EZ signals are shown as follows.

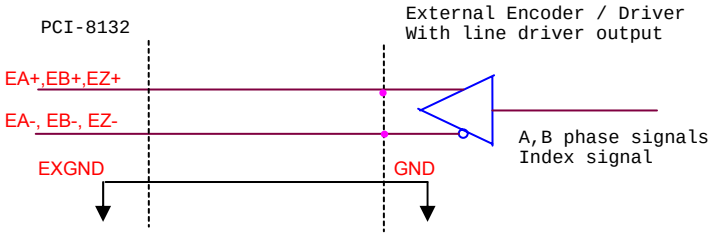


Please note that the voltage across every differential pair of encoder input signals (EA+, EA-), (EB+, EB-) and (EZ+, EZ-) should be **at least 3.5V** or higher. Therefore, you have to take care of the driving capability when connecting with the encoder feedback or motor driver feedback. The differential signal pairs will be converted to digital signal EA, EB and EZ to connect to PCL5023 ASIC.

Here are two examples of connecting the input signals with the external circuits. The input circuits can connect to the encoder or motor driver, which are equipped with: (1) differential line driver or (2) open collector output.

♦ **Connection to Line Driver Output**

To drive the PCI-8132 encoder input, the driver output must provide at least **3.5V** across the differential pairs with at least **6 mA** driving capability. The ground level of the two sides must be tight together too.

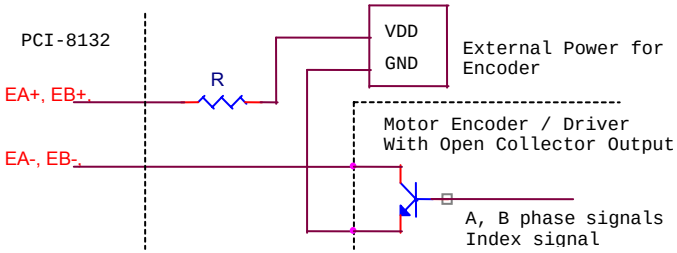


♦ **Connection to Open Collector Output**

To connect with open collector output, an external power supply is necessary. Some motor drivers also provide the power source. The connection between PCI-8132, encoder, and the power supply is shown in the following diagram. Please note that the external current limit resistor R is necessary to protect the PCI-8132 input circuit. The following table lists the suggested resistor value according to the encoder power supply.

Encoder Power(VDD)	External Resistor R
+5V	0 Ω (None)
+12V	1.8kΩ
+24V	4.3kΩ

If=6mA max.



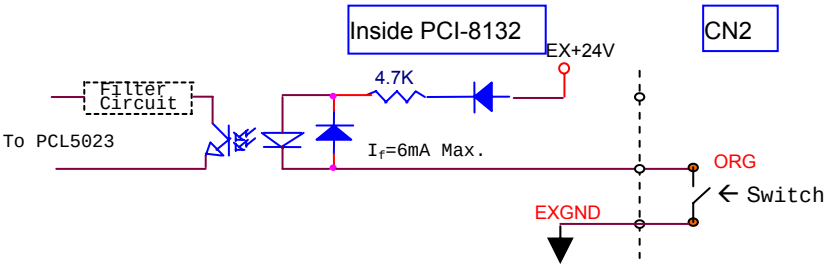
For more detail operation of the encoder feedback signals, please refer to setcion 4.4.

3.3 Origin Signal ORG

The origin signals (ORG1~ORG2) are used as input signals for origin of the mechanism. The following table lists the relative signal name, pin number, and the axis number.

CN2 Pin No	Signal Name	Axis #
41	ORG1	①
47	ORG2	②

The input circuits of the ORG signals are shown as following. Usually, a limit switch is used to indicate the origin of one axis. The specifications of the limit switches should with contact capacity of +24V, 6mA minimum. An internal filter circuit is used to filter out the high frequency spike, which may cause wrong operation.



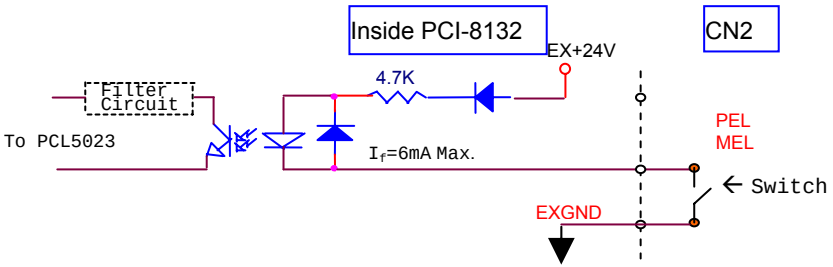
When the motion controller is operated at the home return mode, the ORG signal is used to stop the control output signals (OUT and DIR). For the detail operation of the ORG, please refer to section 4.3.3.

3.4 End-Limit Signals PEL and MEL

There are two end-limit signals PEL and MEL for one axis. PEL indicates end limit signal in plus direction and MEL indicates end limit signal in minus direction. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #	CN2 Pin No	Signal Name	Axis #
37	PEL1	①	43	PEL2	②
38	MEL1	①	44	MEL2	②

The signals connection and relative circuit diagram is shown in the following diagram. The external limit switches featuring a contact capacity of +24V, 6mA minimum. You can use either 'A-type' (normal open) contact switch or 'B-type' (normal closed) contact switch by setting the DIP switch S1. The PCI-8132 is delivered with all bits of S1 set to OFF, refer to section 2.10. For the details of the EL operation, please refer to section 4.3.2.

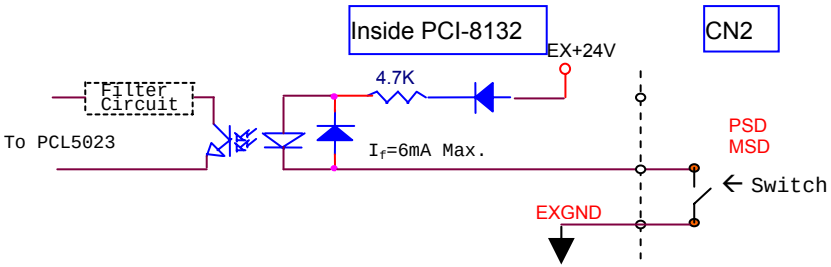


3.5 Ramping-down Signals PSD and MSD

There are two ramping-down (Slow-Down) signals PSD and MSD for one axis. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
39	PSD1	①
40	MSD1	①
45	PSD2	②
46	MSD2	②

The signals connection and relative circuit diagram is shown in the following diagram. Usually, limit switches are used to generate the slow-down signals to make motor operating in a slower speed. For more details of the SD operation, please refer to section 4.3.1.

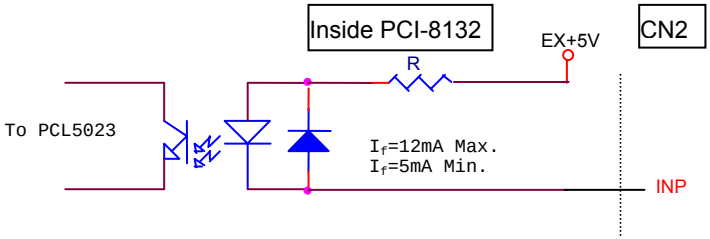


3.6 In-position Signal INP

The in-position signals INP from the servo motor driver indicate the deviation error is zero, that is the servo position error is zero. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
10	INP1	①
28	INP2	②

The input circuit of the INP signals are shown in the following diagram.



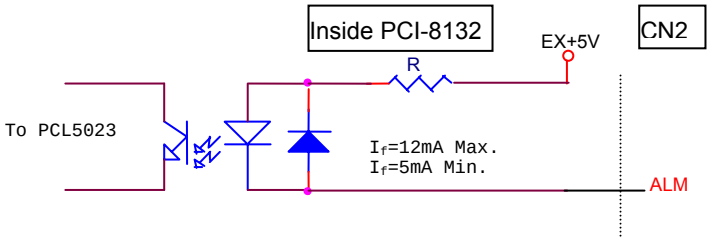
The in-position signals are usually from servomotor drivers, which usually provide open collector output signals. The external circuit must provide at least 5 mA current sink capability to drive the INP signal active. For more details of the INP signal operating, please refer to section 4.2.1.

3.7 Alarm Signal ALM

The alarm signal ALM is used to indicate the alarm status from the servo driver. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
9	ALM1	①
27	ALM2	②

The input circuit of alarm circuit is shown in the following diagram. The ALM signals are usually from servomotor drivers, which usually provide open collector output signals. The external circuit must provide at least 5 mA current sink capability to drive the ALM signal active. For more details of the ALM operation, please refer to section 4.2.2.



3.8 Deviation Counter Clear Signal ERC

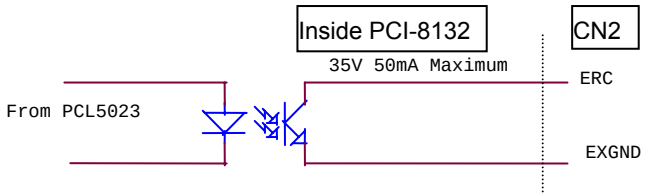
The deviation counter clear signal (ERC) is active in the following 4 situations:

(1) home return is complete;
(2) the end-limit switch is active;
(3) an alarm signal stops OUT and DIR signals;
(4) an emergency stop command is issued by software (operator).

The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
8	ERC1	①
26	ERC2	②

The ERC signal is used to clear the deviation counter of servomotor driver. The ERC output circuit is in the open collector with maximum 35 V external power at 50mA driving capability. For more details of the ERC operation, please refer to section 4.2.3.

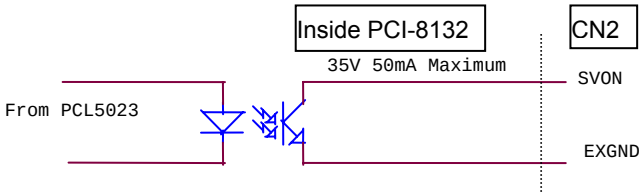


3.9 General-purpose Signal SVON

The SVON signals can be used as servomotor-on control or general-purpose output signals. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
7	SVON1	①
25	SVON2	②

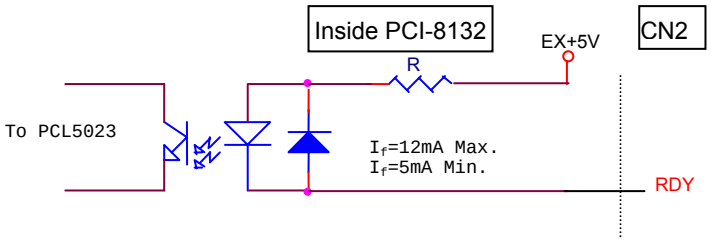
The output circuit of SVON signal is shown in the following diagram.



3.10 General-purpose Signal RDY

The RDY signals can be used as motor driver ready input or general-purpose input signals. The relative signal name, pin number and axis number are shown in the following table.

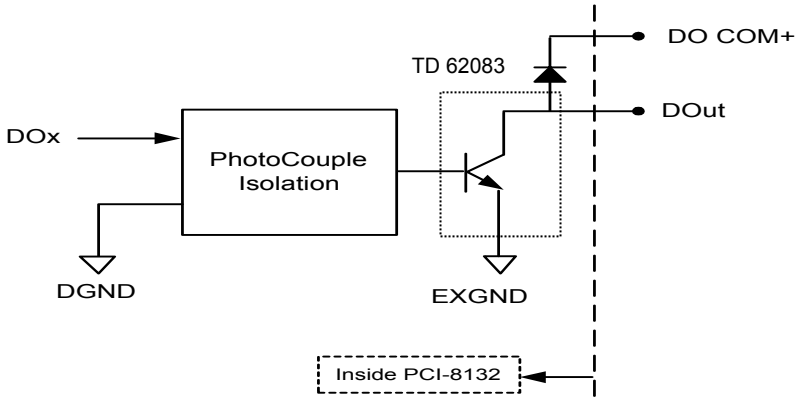
CN2 Pin No	Signal Name	Axis #
11	RDY1	①
29	RDY2	②



The input circuit of RDY signal is shown in the following diagram

3.11 Isolated Digital Output DOx

The connection of isolated-digital output is shown as following diagram. When the isolated digital output goes to high, the sink current will be from external Dout supplied voltage. Each transistor on TD62083 is at OFF State when reset.

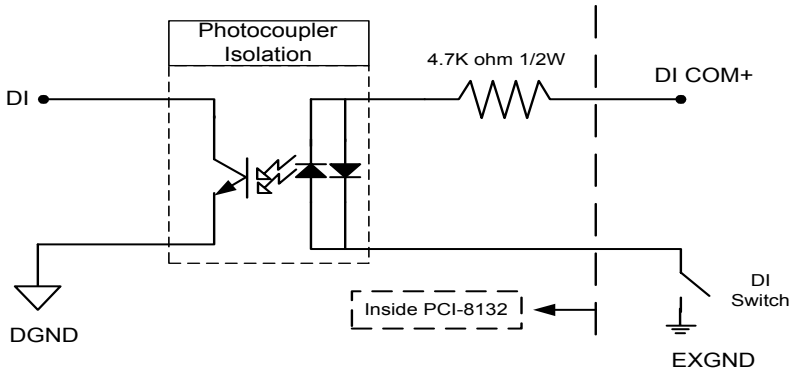


Spec. of TD62083

- Output sustaning voltage: 50V
- Output Current: 123 mA/ch (Duty=50%), 500 mA/ch(MAX.)
- Clamp Diode Reverse Voltage: 50V
- Clamp Diode Forward Current: 500mA
- Power Dissipation: 1.47W (maximum)

3.12 Isolated Digital Input DIx

The isolated digital input is open collector transistor structure. The Input voltage range from 5V to 24V and input resister is 4.7K (1/2W). The connection between outside signal is shown bellow. Maximum forward current through the diode of photocoupler is +/- 50mA

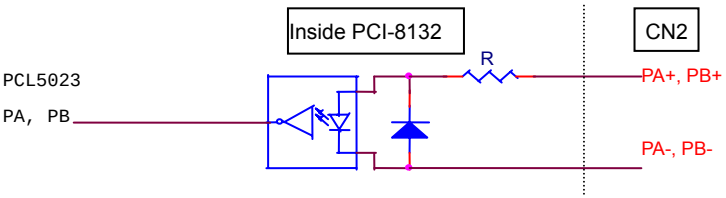


3.13 Pulser Input Signals PA and PB

The PCI-8132 can accept the input signals from pulser signals through the following pins of connector CN2. The pulser's behavior is as an encoder. The signals are usually used as generate the position information which guide the motor to follow.

CN2 Pin No	Signal Name
91	PA+
92	PA-
93	PB+
94	PB-

PA and PB pins of connector CN2 are directly connected to PA and PB pins of PCL5023. The interfac circuits are shown as follows.

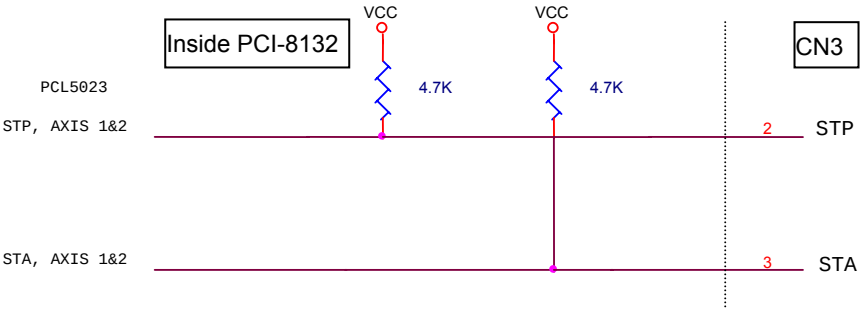


If the signal voltage of pulser is not +5V or if the pulser is distantly placed, it is recommended to put a photo coupler or line driver in between.

3.14 Simultaneously Start/Stop Signals STA and STP

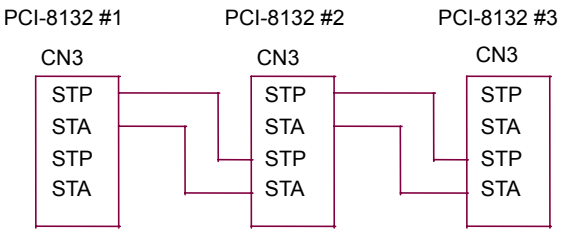
The PCI-8132 provides the STA and STP signals, which enable simultaneous start/stop of motions on multiple axes. The STA and STP signals are on the CN3.

On one card, two PCL5023 chips provide two sets of STA and STP signals. The following diagram shows the on-board circuits. The STA and STP signals of the two axes are tight together respectively.

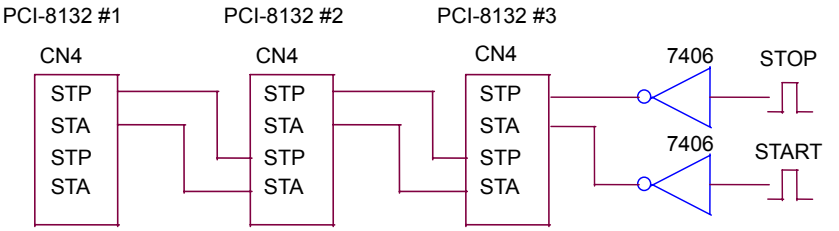


The STP and STA signals are both input and output signal. To operate the simultaneously start and stop action, both software control and external control are possible. By the software control, the signals can be generated from any one of the PCL5023, and other chip will start and stop simultaneously if proper programmed. You can also use an external open collector or switch to drive the STA/STP signals for simultaneous start/stop.

If there are two or more PCI-8132 cards, cascade CN3 connectors of all cards for simultaneous start/stop control on all concerned axes is possible. In this case, connect CN3 as follows.



To let an external signal to initiate simultaneous start/stop, connect the 7406 (open collector) or the equivalent circuit as follows.



3.15 Daughter Board Connector

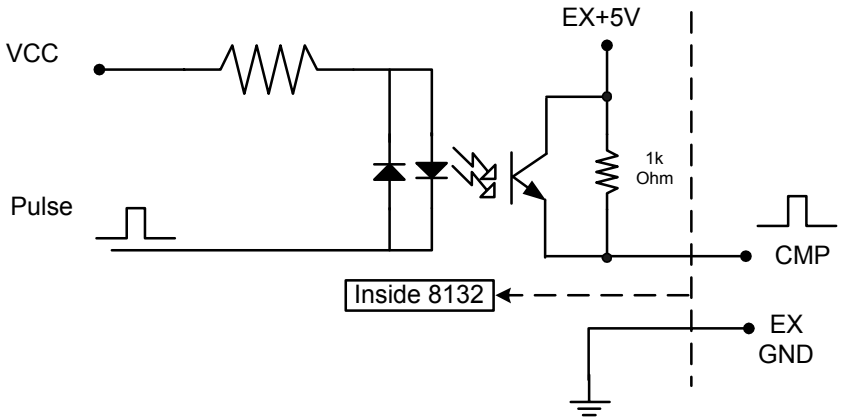
The CN2 connector of PCI-8132 can be connected with DIN-100S, including a cable ACL-102100 (a 100-pin SCSI-II cable). DIN-100S is a general purpose DIN-socket with 100-pin SCSI-II connector. It has easily wiring screw terimal and easily installation DIN socket that can be mounted on DIN-rails

Please check the NuDAQ catalog by ADLINK for further information of DIN-100S.

3.16 Comparison Output CMP1 and CMP2

The PCI-8132 provides two pins for position compare trigger output. The pulse width of this trigger is 100 micro seconds for most industrial CCD camera. The pin assignment and wiring are as follows:

CN2 Pin No	Signal Name	Axis #
96	CMP1	①
97	CMP2	②



4

Operation Theorem

This chapter describes the detail operation of the PCI-8132 card. Contents of the following sections are as following.

Section 4.1: The motion control modes
Section 4.2: The motor driver interface (INP, ERC, ALM, SVON, RDY)
Section 4.3: The limit switch interface and I/O status (SD, EL, ORG)
Section 4.4: The encoder feedback signals (EA, EB, EZ)
Section 4.5: Multiple PCI-8132 cards operation.
Section 4.6: Change Speed on the Fly
Section 4.7: Position Comparison
Section 4.8: Interrupt Control

4.1 Motion Control Modes

In this section, the pulse output signals' configurations, and the following motion control modes are described.

- Constant velocity motion for one axis
- Trapezoidal motion for one axis
- S-Curve profile motion for one axis
- Linear / Circular interpolation for two axes
- Home return mode for one axis
- Manual pulser mode for one axis

4.1.1 Pulse Command Output

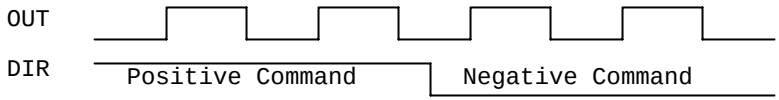
The PCI-8132 uses pulse command to control the servo / stepper motors via the drivers. The pulse command consists of two signals: OUT and DIR. There are two command types: (1) single pulse output mode (OUT/DIR); and (2) dual pulse output mode (CW/CCW type pulse output). The software function: set_pls_outmode() is used to program the pulse command type. The modes vs. signal type of OUT and DIR pins are as following table:

Mode	Output of OUT pin	Output of DIR pin
Dual pulse output	Pulse signal in plus (or CW) direction	Pulse signal in minus (or CCW) direction
Single pulse output	Pulse signal	Direction signal (level)

The interface characteristics of these signals could be differential line driver or open collector output. Please refer to section 3.1 for the jumper setting of signal types.

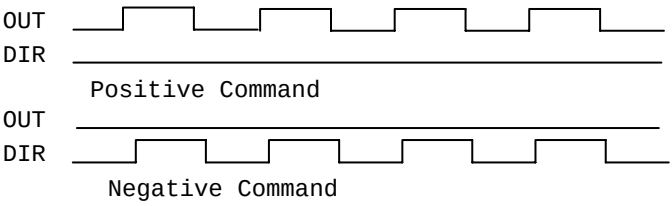
- **Single Pulse Output Mode(OUT/DIR Mode)**

In this mode, the OUT signal is represent the pulse (position or velocity) command. The numbers of OUT pulse represent the motion command for relative "distance" or "position", the frequency of the OUT pulse represents the command for "speed" or "velocity". The DIR signal represents direction command of the positive (+) or negative (-). This mode is the most common used mode. The following diagram shows the output waveform.



- **Dual Pulse Output Mode(CW/CCW Mode)**

In this mode, the waveform of the OUT and DIR pins represents CW (clockwise) and CCW (counter clockwise) pulse output respectively. Pulses output from CW pin makes motor move in positive direction, whereas pulse output from CCW pin makes motor move in negative direction. The following diagram shows the output waveform of positive (plus,+) command and negative (minus,-) command.

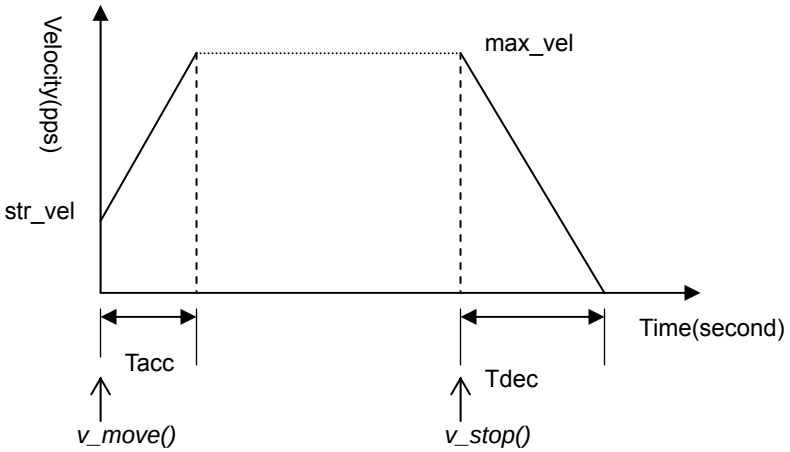


♦ **Relative Function:**
_8132_set_pls_optmode(): Refer to section 6.4

4.1.2 Constant Velocity Motion

This mode is used to operate one axis motor at constant velocity motion. The output pulse accelerates from a starting velocity (str_vel) to the specified constant velocity (max_vel). The ***_8132_*v_move()** function is used to accelerate constantly while the ***_8132_*sv_move()** function is to accelerate according to S-curve (constant jerk). The pulse output rate will keep at maximum velocity until another velocity command is set or stop command is issued. The ***_8132_*v_change()** is used to change speed during moving. The ***_8132_v_stop()*** function is used to decelerate the motion to zero velocity (stop). The velocity profile is shown as following. **Note** that v_stop() function can be also be applied to stop outputting command pulses during **Preset Mode** (both trapezoidal and S-curve Motion) , **Home Mode** or **Manual Pulser Mode** operations**.**

♦ **Relative Functions:**
_8132_v_move( ), _8132_v_stop( ), _8132_sv_move(): Refer to section 5.5



4.1.3 Trapezoidal Motion

This mode is used to move one axis motor to a specified position (or distance) with a trapezoidal velocity profile. Single axis is controlled from point to point. An absolute or relative motion can be performed. In absolute mode, the target position is assigned. In relative mode, the target displacement is assigned. In both absolute and relative mode, the acceleration and the deceleration can be different. The **_8132_motion_done()** function is used to check whether the movement is complete.

The following diagram shows the trapezoidal profile. There are 9 relative functions. In the **_8132_a_move()**,**_8132_ta_move()** and **_8132_start_a_move()**,**_8132_start_ta_move()** functions, the absolute target position must be given in the unit of pulse. The physical length or angle of one movement is dependent on the motor driver and the mechanism (includes the motor). Since absolute move mode needs the information of current actual position, so “External encoder feedback (EA, EB pins)” must be enabled in **_8132_set_cnt_src()** function. And the ratio between command pulses and external feedback pulse input must be appropriately set by **_8132_set_move_ratio()** function.

In the **_8132_r_move()**,**_8132_t_move()** and **_8132_start_r_move()**, **_8132_start_t_move()** functions, the relative displacement must be given in the unit of pulse. Unsymmetrical trapezoidal velocity profile (Tacc is not equal Tdec) can be specified in **_8132_ta_move()** and **_8132_t_move()** functions; where symmetrical profile (Tacc = Tdec) can be specified in **_8132_a_move()** and **_8132_r_move()** functions

The str_vel and max_vel parameters are given in the unit of pulse per second (pps). The Tacc and Tdec parameters are given in the unit of second represent accel./decel. time respectively. You have to know the physical meaning of “one movement” to calculate the physical value of the relative velocity or acceleration parameters. The following formula gives the basic relationship between these parameters.

```
max_vel = str_vel + accel*Tacc;
str_vel = max_vel + decel *Tdec;
```

where accel/decel represents the acceleration/deceleration rate in unit of pps/sec. The area inside the trapezoidal profile represents the moving distance.

The unit of velocity setting is pulses per second (pps). Usually, the unit of velocity in the manual of motor or driver is in rounds per minute (rpm). A simple conversion is necessary to match between these two units. Here we use a example to illustrate the conversion.

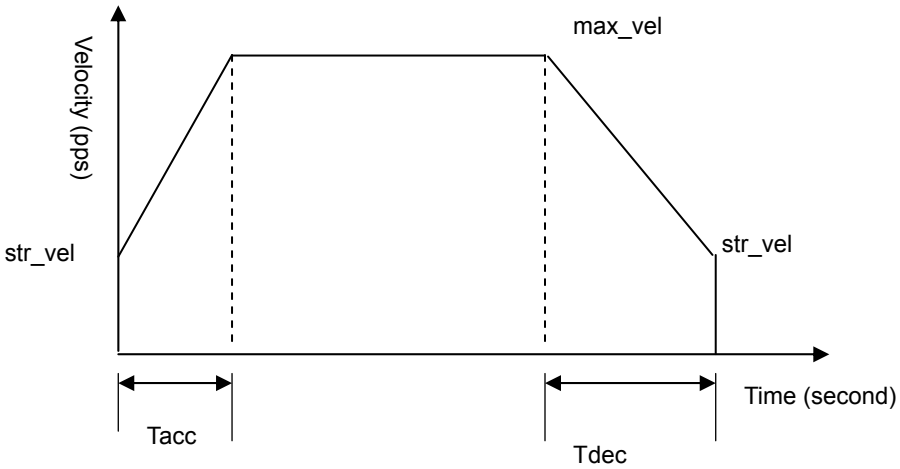
For example:

A servo motor with a AB phase encoder is used for a X-Y table. The resolution of encoder is 2000 counts per phase. The maximum rotating speed of motor is designed to be 3600 rpm. What is the maximum pulse command output frequency that you have to set on PCI-8132?

Answer:

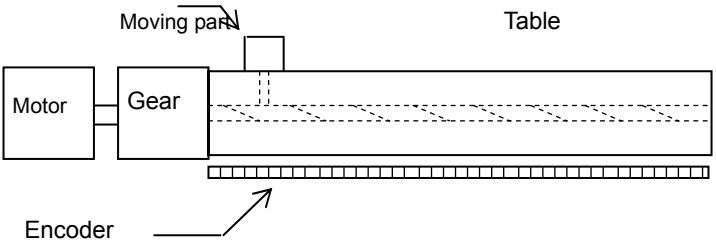
max_vel = 3600/60*2000*4
= 48000pps

The reason why *4 is because there are four states per AB phase (See Figures in Section 4.4).



Usually, the axes need to set the move ratio if their mechanical resolution is different from the resolution of command pulse. For example, if an incremental type encoder is mounted on the working table to measure the actual position of moving part. A servomotor is used to drive the moving part through a gear mechanism. The gear mechanism is used to convert the rotating motion of motor into linear motion.(see the following diagram). If the resolution of motor is 8000 pulses/round. The resolution of gear mechanism is 100 mm/round.(i.e., part moves 100 mm if motor turns one round). Then the resolution of command pulse will be 80 pulses/mm. The resolution of encoder mounting on the table is 200 pulses/mm. Then users have to set the move ratio as 200/80=2.5 by the function:

♦ **_8132_set_move_ratio(axis, 2.5);**



If this ratio is not set before issuing the start moving command, it will cause problems when running in “Absolute Mode”. Because the PCI-8132 can’t recognize the actual absolute position during motion.

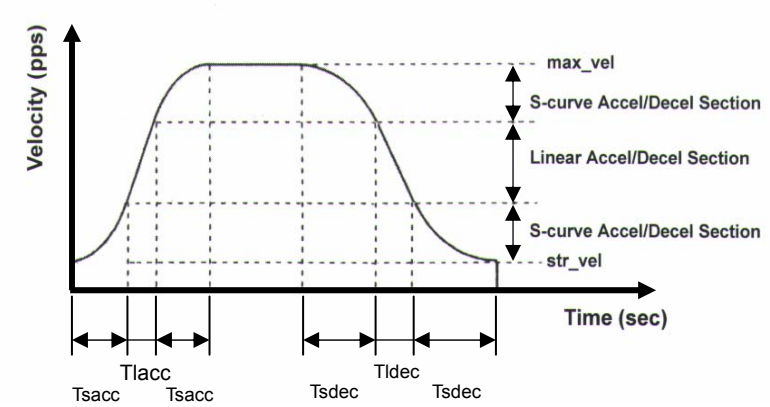
♦ **Relative Functions:**
_8132_a_move(), _8132_r_move(), _8132_t_move(), _8132_ta_move(), _8132_start_a_move(), _8132_start_r_move(), _8132_start_t_move(), _8132_start_ta_move() ***Refer to section 6.6.***
_8132_motion_done(): Refer to section 6.13.
_8132_set_cnt_src(): Refer to section 6.4.
_8132_set_move_ratio(): Refer to section 6.10.

4.1.4 S-curve Profile Motion

This mode is used to move one axis motor to a specified position (or distance) with a S-curve velocity profile. S-curve acceleration profiles are useful for both stepper and servo motors. The smooth transitions between the start of the acceleration ramp and the transition to the constant velocity produce less wear and tear than a trapezoidal profile motion. The smoother performance increases the life of the motors and mechanics of a system.

There are several parameters needed to be set in order to make a S-curve move. They are:

pos: target position in absolute mode;
dist : moving distance in relative mode;
str_vel : specify the start velocity;
max_vel : specify the maximum velocity;
Tlacc : specify the time for linear acceleration section (constant acceleration).
Tsacc : specify the time for S-curve acceleration section (constant jerk).
Tldec : specify the time for linear deceleration section (constant deceleration).
Tsdec : specify the time for S-curve deceleration section (constant jerk).



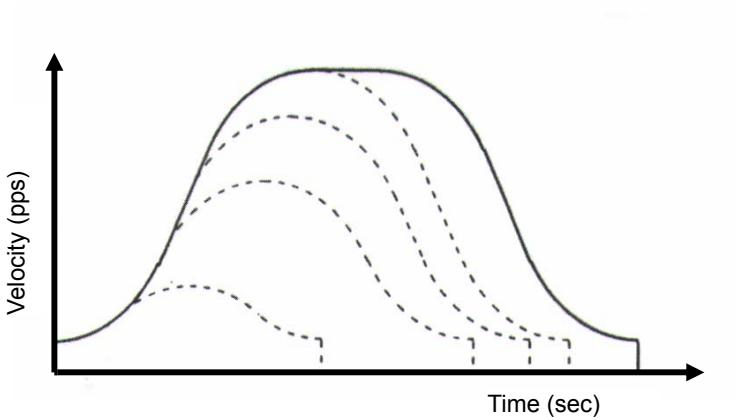
Total time of acceleration is : Tlacc+2Tsacc. The following formula gives the basic relationship between these parameters.

```
max_vel = str_vel + accel*(Tlacc+Tsacc);
str_vel   = max_vel + decel *(Tldec+Tsdec);
accel = Tsacc * jerk1;
decel = Tsdec * jerk2;
```

where accel/decel represents the acceleration/deceleration rate at linear accel./decel. section and are in unit of pps/sec. jerk1, jerk2 are in unit of pps/sec^2. The minimum value for setting time of accel./decel. should be 0.

The S-curve profile motion functions are designed to always produce smooth motion. If the time for linear/S-Curve acceleration parameters combined with the final position don't allow an axis to reach the maximum velocity(i.e.: the moving distance is too small to reach max_vel), the maximum velocity is automatically lowered and

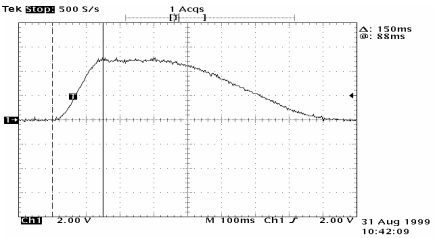
smooth accel./decel. is made (see the following Figure). This means that with moves that don't reach maximum velocity may cause longer than expected move times. In such a case, the smaller the moving distance, the shorter the linear accel./decel. section becomes and the S-curve section is not reduced unless the linear section is decreased to 0.



The following two graphs show the results of experiments after executing the unsymmetrical absolute S-curve motion command. Graph1 is the typical result. of S-curve velocity profile. Graph2 is obtained when the amount of command pulses is failed to let the velocity reach the designated maximum velocity. The PCI-8132 automatically lower the maximum velocity thus provide a smooth velocity profile.

Command of Graph1:

start_tas_move(axis, 500000, 100, 1000000, 0.05, 0.05, 0.2, 0.2);

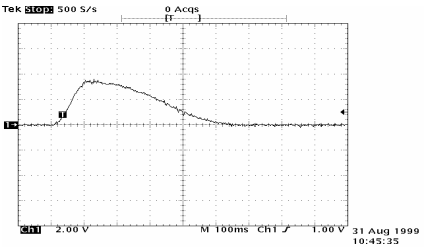


The total accelerating time = 0.05+2*0.05 = 0.15 (second).

Total decelerating time = 0.2+2*0.2 = 0.6 (second).

Command of Graph2:

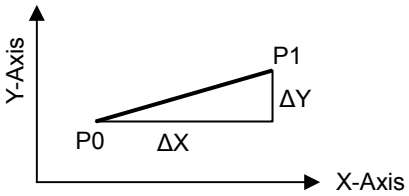
start_tas_move(axis, 200000, 100, 1000000, 0.05, 0.05, 0.2, 0.2);



- **Relative Functions:**
  _8132_s_move(),_8132_rs_move(),_8132_tas_move(), _8132_start_s_move(),_8132_start_rs_move(),_8132_start_tas_move() Refer to section 6.7
  _8132_motion_done(): Refer to section 6.13

4.1.5 Linear Interpolated Motion

In this mode, two axes (“X and Y” or “Z and U” axes) is controlled by linear interpolation or circular interpolation by designating the number of pulses respectively. “Interpolation between two axes” means the two axes start simultaneously, and reach their ending points at the same time. For example, in the Figure below, we want to move the axes from P0 to P1, and hope the two axes start and stop simultaneously at a period of time Δt. Then the moving speed along X-axis and Y-axis will beΔX/Δt., ΔY/Δt. respectively.



The axis with larger numbers of moving pulses is the main axis, and the other axis is the secondary axis. When both axes are set at the same amount of pulses, the ‘X’ is the main axis. The speed relation between main and secondary axes is as follows:

Composite Speed = Speed of main axis x

$$\frac{\sqrt{(\text{Set number of pulses for main axis})^2 + (\text{Set number of pulses for slave axis})^2}}{\text{Set number of pulses for main axis}}$$

$$\text{or} = \text{Speed of main axis} \times \sqrt{1+\left(\frac{\text{Set number of pulses for slave axis}}{\text{Set number of pulses for main axis}}\right)^2}$$

- **Relative Functions:**
  _8132_move_xy(),_8132_start_move_xy(): Refer to section 6.9
  _8132_set_move_speed(),_8132_set_move_accel() ,_8132_set_move_ratio(): Refer to section 6.10

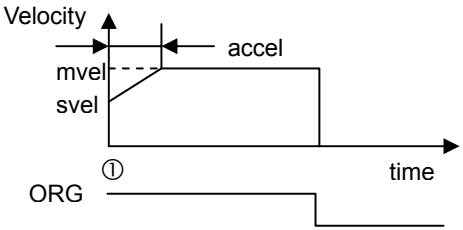
4.1.6 Home Return Mode

In this mode, you can let the PCI-8132 output pulses until the conditions to complete the home return is satisfied after writing the **_8132_home_move()** command. Finish of home return can be checked by **_8132_motion_done()** function. Or you can check finish of home return accompanied with the interrupt function by setting bit 5 of ***int_factor*** to 1 in **_8132_*set_int_factor()*** function.

Moving direction of motors in this mode is determined by the sign of velocity parameter in **_8132_home_move()** function. A **_8132_v_stop()** command during returning home can stop OUT and DIR from outputting pulses.

Before writing **_8132_home_move()** command, configuration must be set by **_8132_set_home_config()** function. . See also Section 4.3.3 for further description. There are total three home return modes can be selected by setting home_mode parameter in **_8132_set_home_config()** function. The meaning of Home_mode will be described as the following:

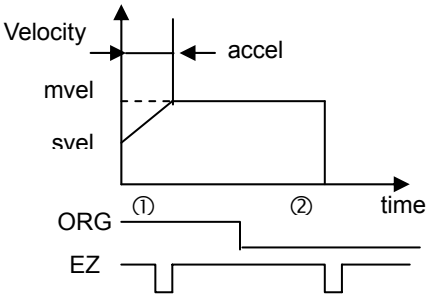
1. Home_mode=0: ORG only, no index signal. The ORG signal immediately stops OUT and DIR pins from outputting pulses to complete the origin return.



① Writing home-move( ) command to begin home return operation

②ORG Signal ON

2. Home_mode=1: both ORG and index signal are useful. The ORG signal lets the PCI-8132 starts to wait for EZ signal and then EZ signal stops OUT and DIR pins from outputting pulses to complete the home return.

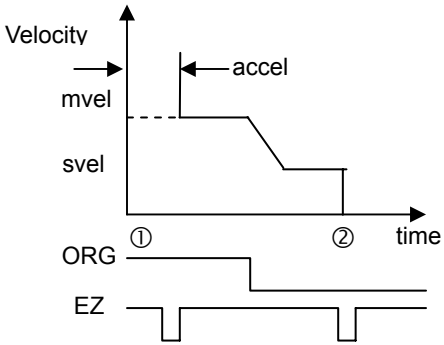


① Writing home-move( ) command to begin home return operation

②ORG Signal ON

③EZ Signal ON

3. Home_mode=2: both ORG and index signal are useful. The ORG signal lets the PCI-8132 decelerate to starting velocity and then EZ signal stops OUT and DIR pins from outputting pulses to complete the home return.



Note: If the starting velocity is zero, the axis will work properly in home mode 2.

♦ **Relative Function:**

_8132_set_home_config(),_8132_home_move(),_8132_v_stop(): Refer to section 6.11

4.1.7 Manual Pulser Mode

For manual operation of a device, you may use a manual pulser such as a rotary encoder. The PCI-8132 can input signals from the pulser and output corresponding pulses from the OUT and DIR pins, thereby allowing you to simplify the external circuit and control the present position of axis. This mode is effective between a **_8132_manu_move()** command is written and a **_8132_v_stop()** command.

The PCI-8132 receives plus and minus pulses (CW/CCW) or 90 degrees phase difference signals(AB phase) from the pulser at PA and PB pins. The 90° phase difference signals can be input through multiplication by 1, 2 or 4. If the AB pahse input mode is selected, the PA and PB signals should be with 90° phase shifted, and the position counting is increasing when the PA signal is leasding the PB signal by 90° phase.

Also, one pulser may be used for 'X' and 'Y' axes while internally distributing the signals appropriately to two axes. To set the input signal modes of pulser, use **_8132_set_manu_iptmode()** function. Then write **_8132_manu_move()** to begin manual operation function. User must write **_8132_v_stop()** command in order to end this function and begins to operate at another mode. User can choose pulse output axis by ***_8132_*set_manu_axis().**

The error input of PA and PB can be used to generate IRQ. The following two situations will be considered as error input of PA and PB signals. (1) The PA and PB signals are changing simultaneously. (2) The input pulser frequency is higher than the maximum output frequency 2.4M pps. Set bit 14 of INT factor will enable the IRQ when error happen.

Maximum moving velocity in this mode can be limited by setting max_vel parameter in **_8132_manu_move()** function.

- **Relative Function:**

_8132_set_manu_iptmode(),_8132_manu_move(),_8132_manu_axis(), _8132_v_stop(): Refer to section 6.12

4.2 Motor Driver Interface

The PCI-8132 provides the INP, ERC and ALM signals for servomotor driver's control interface. The INP and ALM are used for feedback the servo driver's status. The ERC is used to reset the servo driver's deviation counter under special conditions.

4.2.1 INP

Usually, servomotor driver with pulse train input has a deviation (position error) counter to detect the deviation between the input pulse command and feedback counter. The driver controls the motion of servomotor to minimize the deviation until it becomes 0. Theoretically, the servomotor operates with some time delay from command pulses. Accordingly, when the pulse generator stops outputting pulses, the servomotor does not stop but keep running until the deviation counter become zero. At this moment, the servo driver sends out the in-position signal (INP) to the pulse generator to indicate the motor stops running.

Usually, the PCI-8132 stops outputting pulses upon completion of outputting designated pulses. But by setting ***inp_enable*** parameter in **_8132_*set_inp_logic()*** function , you can delay the completion of operation to the time when the INP signal is turned on. Status of **_8132_motion_done()** and INT signal are also delayed. That is, when performing under position control mode, the completion of **_8132_start_a_move()**, **_8132_start_r_move()**, **start_s_move()**… functions are delayed until INP signal is turned ON.

However, EL or ALM signal or the completion of home return does not cause the INP signal to delay the timing of completion. The INP signal may be a pulse signal, of which the shortest width is 5 micro seconds.

The in-position function can be enable or disable. The input logic polarity isalso programmable by software function: **_8132_set_inp_logic()**. The signal status can be monitored by software function: **_8132_get_io_status()**.

4.2.2 ALM

The ALM pin receives the alarm signal output from the servo driver. The signal immediately stops the PCI-8132 from generating pulses or stops it after deceleration. If the ALM signal is in the ON status at the start, the PCI-8132 outputs the INT signal without generating any command pulse. The ALM signal may be a pulse signal, of which the shortest width is a time length of 5 micro seconds.

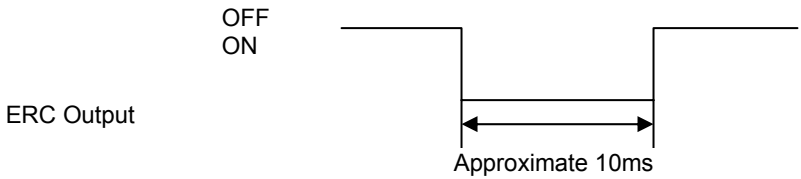
You can change the input logic by ***_8132_set_alm_logic()*** function. Whether or not the PCI-8132 is generating pulses, the ALM signal lets it output the INT signal.. The ALM status can be monitored by software function: ***_8132_get_io_status()***. The ALM signal can generate IRQ by setting the bit 2 of INT. factor in software function: **_8132_set_int_factor().**

4.2.3 ERC

The deviation counter clear signal is inserted in the following 4 situations:

1. home return is complete;
2. the end-limit switch is active;
3. an alarm signal stops OUT and DIR signals;
4. an emergency stop command is issued by software operator.

Since the servomotor operates with some delay from pulse generated from the PCI-8132, it keeps operating by responding to the position error remaining in the deviation counter of the driver if the ±EL signal or the completion of home return stops the PCL5023 from outputting pulses. The ERC signal allows you to immediately stop the servomotor by resetting the deviation counter to zero. The ERC signal is output as an one-shot signal. The pulsewidth is a time length of 10ms. The ERC signal will automatically output when ±EL signals, ALM signal is turned on to immediately stop the servomotor. User can set the ERC pin output enable/disable by ***_8132_set_erc_enable()*** function. ERC pin output is set output enabled when initializing.



4.3 The Limit Switch Interface and I/O Status

In this section, the following I/O signals' operations are described.

- ±SD: Ramping Down sensor
- ±EL: End-limit sensor
- ORG: Origin position
- SVON and RDY

I/O status readback

In any operation mode, if an ±EL signal is active during moving condition, it will cause PCI-8132 to stop output pulses automatically. If an SD signal is active during moving condition, it will cause PCI-8132 to decelerate.

4.3.1 SD

The ramping-down signals are used to slow-down the control output signals (OUT and DIR) when it is active. The signals are very useful to protect the mechanism moving under high speed toward the mechanism limit. PSD indicates ramping-sown signal in plus (+) direction and MSD indicates ramping-down signal in minus (-) direction.

During varied speed operation in the home return mode or continuous operation mode, the ramping-down signal in the moving direction lets the output control signals (OUT and DIR) ramp down to the pre-setting starting velocity.

The ramping-down function can be enable or disable by software function: **_8132_set_sd_logic()**. The input logic polarity, level operation mode, or latched input mode can also be set by this function. The signals status can be monitored by **_8132_get_io_status()**.

4.3.2 EL

The end-limit signals are used to stop the control output signals (OUT and DIR) when the end-limit is active. PEL signal indicates end-limit in positive (plus) direction. MEL signal indicates end-limit in negative (minus) direction. When the output pulse signals (OUT and DIR) are toward positive direction, the pulse train will be immediately stopped when the PEL signal is inserted, while the MEL signal is meaningless in this case, and vise versa. When the PEL is inserted and the output pulse is fully stop, only the negative (minus) direction output pulse can be generated for moving the motor to negative (minus) direction.

The end-limit signals can be used to generate the IRQ by setting the bit 0 of INT. factor in software function: **_8132_set_int_factor()**.

You can use either 'a' contact switch or 'b' contact switch by setting the dip switch S1. The PCI-8132 is delivered from the factory with all bits of S1 set to OFF.

The signal status can be monitored by software function: **_8132_get_ io_status().**

4.3.3 ORG

When the motion controller is operated at the home return mode, the ORG signal is used to stop the control output signals (OUT and DIR).

There are three home return modes, you can select one of them by setting "*home_mode*" argument in software function: set_home_config(). Note that if home_mode=1 or 2, the ORG signal must be ON or latched during the EZ signal is inserted (EZ=0). The logic polarity of the ORG signal, level input or latched input mode are selectable by software function: **_8132_set_ home_config()**.

After setting the configuration of home return mode by **_8132_set _home_config()**, a home_move() command can perform the home return function.

The ORG signal can also generate IRQ signal by setting the bit 5 of interrupt reason register (or INT. factor) in software function: **_8132_set_int_factor()**.

4.3.4 SVON and RDY

The SVON signals are controlled by software function: **_8132_Set_SVON()**. The function set the logic of AP0 (SVON) of PCL5023. The signal status of SVON pins can be monitored by software function: **_8132_get_io_status()**.

RDY pins are dedicated for digital input use The status of this signal can be monitored by software function get_io_status(). RDY pin is interfaced with AP3 pin of PCL5023 through a photocoulpe. The RDY signal can also generate IRQ signal by setting the bit 23 of INT. factor in software function: set_int_factor(). Note that interrupt is generated when AP3 from high to low.

4.4 The Encoder Feedback Signals (EA, EB, EZ)

The PCI-8132 has a 28-bits binary up/down counter for managing the present position for each axis. The counter counts signals input from EA and EB pins.

It can accept 2 kinds of pulse input.: (1). plus and minus pulses input(CW/CCW mode); (2). 90° phase difference signals(AB phase mode). 90° phase difference signals may be selected to be multiplied by a factor of 1,2 or 4. 4x AB phase mode is the most commonly used for incremental encoder input. For example, if a rotary encoder has 2000 pulses per phase (A or B phase), then the value read from the counter will be 8000 pulses per turn or –8000 pulses per turn depends on its turning direction. These input modes can be selected by ***_8132_set_pls_iptmode()*** function.

To enable the counters counting pulses input from (EA, EB) pins, set "***cnt_src***" parameter of software function ***_8132_set_cnt_src()*** to 1.

- **Plus and Minus Pulses Input Mode(CW/CCW Mode)**

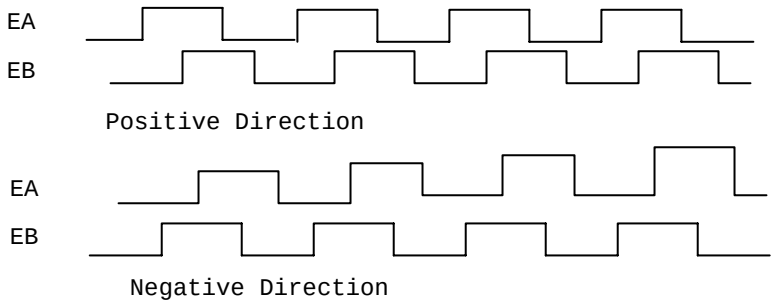
The pattern of pulses in this mode is the same as ***Dual Pulse Output Mode*** in Pulse Command Output section, expect that the input pins are EA and EB.

In this mode, pulse from EA causes the counter to count up, whereas EB caused the counter to count down.

- **90° phase difference signals Input Mode(AB phase Mode)**

In this mode, the EA signal is 90° phase leading or lagging in comparison with EB signal. Where "lead" or "lag' of phase difference between two signals is caused by the turning direction of motors. The up/down counter counts up when the phase of EA signal leads the phase of EB signal.

The following diagram shows the waveform.



The encoder error interrupt is provided to detect abnormal situation. Simultaneously changing of EA and EB signals will cause an encoder error. If bit #14 of the interrupt factor register (INT factor) is set as 1, the IRQ will be generated when detect encoder error during operation.

The index inputs (EZ) signals of the encoders are used as the "ZERO" index. This signal is common on most of the rotational motors. EZ can be used to define the absolute position of the mechanism. The input logic polarity of the EZ signals is programmable by software function ***_8132_set_home_config().*** The EZ signals status of the four axis can be monitored by ***_8132_get_io_status()***.

- **Relative Function:**

_8132_set_cnt_src(),_8132_set_pls_iptmode(): Refer to section 6.4

4.5 Multiple PCI-8132 Cards Operation

The software fuction library support maximum up to 12 PCI-8132 Cards, that means maximum up to 24 axes of motors can be controlled. Since PCI-8132 has the characteristic of Plug-and-Play, users do not have to care about setting the Based address and IRQ level of cards. They are automatically assigned by the BIOS of system when booting up. Users can utilize Motion Creator to check if the plugged PCI-8132 cards are successfully installed and see the Baseaddress and IRQ level assigned by BIOS.

One thing needed to be noticed by users is to identify the card number of PCI-8132 when multiple cards are applied. The card number of one PCI-8132 depends on the locations on the PCI slots. They are numbered either from left to right or right to left on the PCI slots. These card numbers will effect the corresponding axis number on the cards. And the axis number is the first argument for most funcions called in the library. So it is important to identify the axis number before writing application programs. For example, if 3 PCI-8132 cards are plugged in the PCI slots. Then the corresponding axis number on each card will be:

Axis No. / Card No.	Axis 1	Axis 2
1	0	1
2	2	3
3	4	5

If we want to accelerate Axis 1 of Card2 from 0 to 10000pps in 0.5sec for Constant Velocity Mode operation. The axis number should be 6. The code on the program will be:

```
_8132_v_move(2, 0, 10000, 0.5);
```

To determine the right card number, Try and Error may be necessary before application. Motion Creator can be utilized to minimize the search time.

For applications needed to move many axes simultaneously on multiple PCI_8132 cards, users should follow the connection diagrams in Section 3.12 to make connections between their CN3 connectors. Several functions illustrated in Section 6.8 may be useful when writing programs for such applications.

- **Relative Function:**

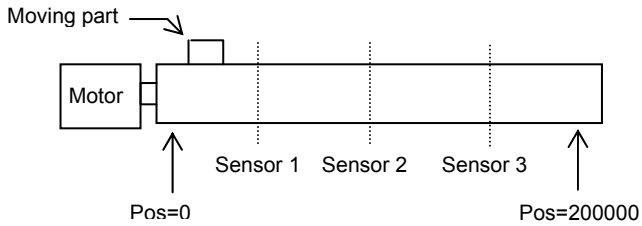
_8132_start_move_all(), _8132_move_all(), _8132_wait_for_all(): Refer to section 6.8

4.6 Change Speed on the Fly

You can change the velocity profile of command pulse ouput during operation by **_8132_*v_change()*** function. This function changes the maximum velocity setting during operation. However, if you operate under "Preset Mode" (like start_a_move(),…), you are not allowed to change the acceleration parameter during operation because the deceleration point is pre-determined. But changing the acceleration parameter when operating under "Constant Velocity Mode" is valid. Changing speed pattern on the fly is valid no matter what you choose "Trapezoidal Velocity Profile" or "S-curve Velocity Profile". Here we use an example of Trapezoidal velocity profile to illustarte this function.

Example: There are 3 speed change sensor during an absolute move for 200000 pulses. Initial maximum speed is 10000pps. Change to 25000pps if Sensor 1 is touched. Change to 50000pps if Sensor 2 is touched. Change to 100000pps if Sensor 3 is touched. Then the code for this application and the resulting velocity profiles are shown below.

User must set ***_8132_*fix_max_speed()** before any PTP motion in order to get the better performance of speed change. The value in this function is the possible maximum speed during the PTP motion.



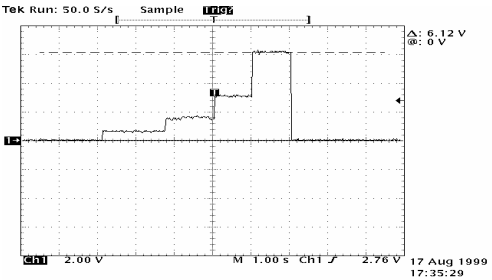

```
#include “pci_8132.h”

_8132_fix_max_speed(axis,100000);
_8132_start_a_move(axis, 200000.0, 1000, 10000, 0.02);

while(!_8132_motion_done(axis))
    {
                    // Get Sensor’s information from other I/O card

  if((Sensor1==High) && (Sensor2==Low) && (Sensor3 == Low))
                _8132_v_change(axis, 25000, 0.02);
  else if((Sensor1==Low) && (Sensor2==High) && (Sensor3 == Low))
                _8132_v_change(axis, 50000, 0.02);
          else if((Sensor1==Low) && (Sensor2==Low) && (Sensor3 ==
High))
                _8132_v_change(axis, 100000, 0.02);
    }
```

Where the informations of three sensors are acquired from other I/O card. And the resulting velocity profile from experiment is shown below.

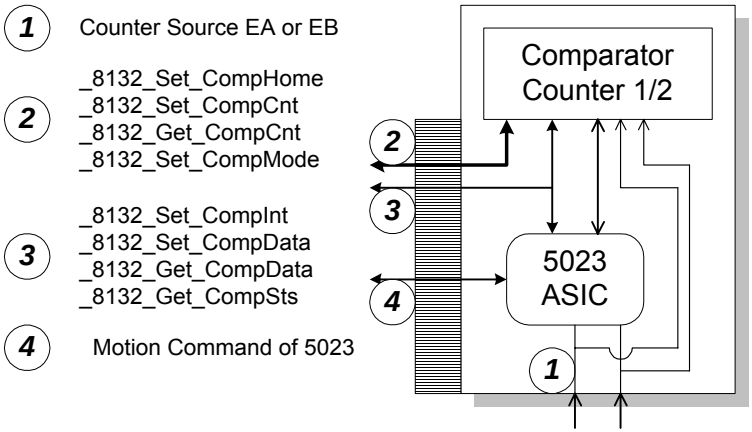


♦ **Relative Function:**

_8132_v_change(), _8132_fix_max_speed(): Refer to section 6.5

4.7 Position Comparison

The position comparison function is fulfilled by the FPGA comparator on board. Please refer to the following figure. The comparator is applied to compare the preset comparison data with the contents of its counter under different modes. These comparison modes consist of different logical comparison (>/=/<) of different counters (1 and/or 2).



To make use of position comparison function the following guidelines will be of much help.

1. ***Decides the comparison mode*** : Use*_8132_*Set_CompMode function and consider the counter source and the comparison conditions .
2. ***Sets the counter initial value***: There are two ways to set the counter
   Directly use *_8132_*Set_CompCnt function to set its value
   Use *_8132_*Set_CompHome to set its value to 0 automatically after homing
3. ***Enables the interrupt function*** : Use *_8132_*Set_CompInt function
4. ***Sets up the desired comparison data***: Use *_8132_* Set_CompData function.
5. ***Gets the status of the comparator :*** Use *_8132_*Get_CompSts
6. ***Sending motion commands*** :After setting up the comparator users can send other moiton control functions eg. start_a_move ,or v_move etc . The comparator will fulfill the comparison function without interfering the CPU.

For user who want to compare multiple data continuously with the comparator

The method of building comparison tables is also provided as shown in the following

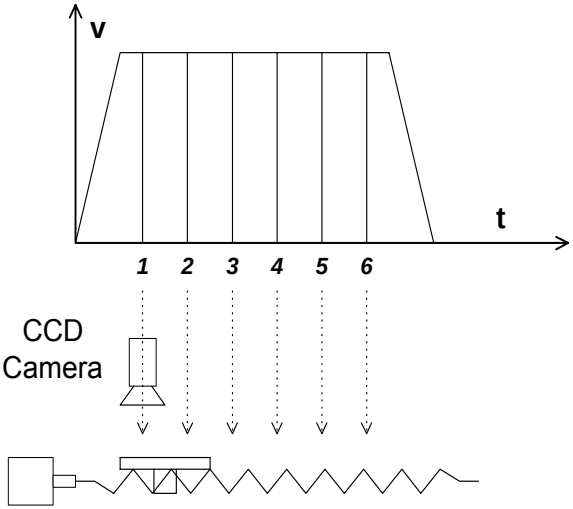
1. U16 _8132_Build_Comp_Table(U16 axis, I32 *table, I16 Size);

   I32 *table: an one dimension array pointer for compare positions
   I16 Size: Total amount of position compare points (Maximum=1024)

2. U16 _8132_Set_Comp_Table(U16 axis, U16 logic);
   U16 logic: enable/disable position compare table
   (0 for disable, 1 for enable)

Here are two examples of using position comparison functions.

The first example is typically in the application of machine vision.



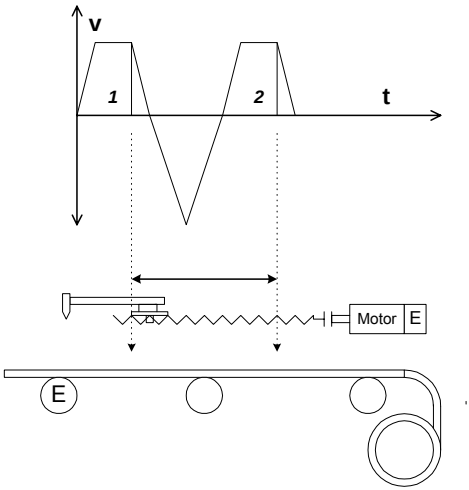
In this application the table is controlled by the motion command and the CCD Camera is controlled by the position comparison output of PCI-8132.

The image of moving object can be get in this way easily.

The example code is shown in the following

```
_8132_Set_CompHome(0);
for( i=0 ; i<6 ; i++)
       CompTable[i] = 10000 + 10000 * i; ' Set Compare Data
    _8132_Build_Comp_Table(0, CompTable, 6);
    _8132_Set_CompMode(0, 0);
    _8132_Set_CompInt( Axis0, 1);
    _8132_Set_Comp_Table( Axis0, 1);
    _8132_start_r_move( Axis0, 80000, 0,10000, 0.5);
```

The second example is a fly-cut application



In this application the cutter is moved forward and backward on the x-axis and the knife is moved up and down by the y-axis. The comparator is used to compare the actual position in x-axis with the encoder feedback on y-axis with the encoder mounted under the belt. I.e.The comparator counter source in this case is the encoder under the belt but not the encoder on the back of the motor. In this application the cutter will cut down when the motor reaches the same speed as the belt and the comparison condition is match.

The comparator in the PCI-8132 generates an interrupt to move the knife down to cut the belt.

The following graph shows the result of position compare trigger output. A compare point table is triggered during a start_a_move() function. The compare table contents 1024 points from 10000 to 112300 with 100 pulses

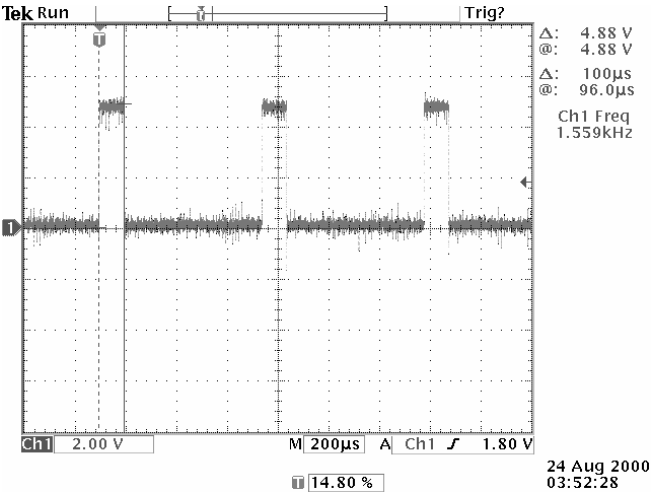
interval. It can be represented as follows:

```
For(i = 0 ; i < 1024; i++ )
        CMP_TBL(i)= 10000+100*i;
```

Once the axis passes by these preset points during a moving function, the corresponding compare output pin will send a pulse with 100 us width to trigger other device to work. The moving command for this example is as follows:

```
start_a_move(AXIS0, 150000, 1000, 160000, 0.2);
```

The maximum command for this function is 160k pps. So the axis takes about 625us to travel 100 pulses long and the width of trigger pulse is about 100us. (The maximum frequency for trigger signal is about 10k)



- **Relative Function:**

_8132_Set_CompHome(),_8132_Build_Comp_Table(),_8132_Set_CompMode(), _8132_Set_CompInt(),_8132_Set_Comp_Table(): Refer to section 6.19

4.8 Interrupt Control

The PCI-8132 motion controller can generate INT signal to host PC according to 13 types of factors, refer to **_8132_set_int_factor()** function for more details.. The INT signal is output when one or more interrupt factors occur on either axis. To judge on which axis the interrupt factors occur, use **_8132_get_int_axis()** function. The interrupt status is not closed until **_8132_get_int_status()** function is performed. There is a little difference between using DOS or Windows 95/NT to perform interrupt control. Users should refer to Section 6.17 for more details. Here we use an example on Windows OS to demonstrate how to perform interrupt control with the function library we provided.

- **Use Thread to deal with Interrupt under Windows NT/95**

In order to detect the interrupt signal from PCI-8132 under Windows NT/95, user must create a thread routine first. Then use APIs provided by PCI-8132 to get the interrupt signal. The sample program is as follows :

Situatuins: Assume that we have one card (2 axes) and want to receive Home Return and Preset Movement Finish interrupt signal from axis 1.

Steps:

1. Define a Global Value to deal with interrupt event

```
HANDLE hEvent[2];
volatile bool ThreadOn;
```

2. In Initializing Section (you must Initialize PCI-8132 properly first), set interrupt types and enable an event for each axis.

```
set_int_factor(1,0x002040);
_8132_Set_INT_Control(0,1);
_8132_INT_Enable(0,&hEvent[0]);
```

Note: For each card, you must assign 2 4-events-array in _8132_INT_Enable function.

3. Define a Global Function (Thread Body). Use WaitForSingleObject() or WaitForMultipleObjects() to wait events. Remenber to reset this event after you get the event.

```
UINT IntThreadProc(LPVOID pParam)
{
  U32 IntSts;

  while(ThreadOn=TRUE)
```

```
    {
        ::WaitForSingleObject(hEvent[1],INFINITE);
        _8132_get_int_status(1,&IntSts);
        ::ResetEvent(hEvent[1]);
    }

    return 0;
}
```

4. Start the thread(Use a boolean value to control the thread's life)
   ThreadOn=TRUE;
   AfxBeginThread(IntThreadProc,GetSafeHwnd(),THREAD_PRIORITY_HIGHEST);

5. Before exit the program, remember to let the thread go to end naturally.
   ThreadOn=FALSE;

For each time when a preset movement or homing of axis 2 is completed, this program will receive a interrupt signal from PCI-8132.

♦ **PCI-8132 Interrupt Service Routine (ISR) with DOS**

A DOS function library is equipped with PCI-8132 for users to develop applications under DOS environment. This library also provide some functions for users to work with ISR. It is highly recommended to write programs according to the following example for applications should work with ISR. Since PCI-bus has the ability to do IRQ sharing when multiple PCI-8132 are applied, each PCI-8132 should have a corresponding ISR. For users who use the library we provide, the names of ISR are fixed, such as: *_8132_isr0(void), _8132_isr1(void)…etc.* The sample program are described as below. It assume two PCI-8132 are plugged on the slot , axis 1 and axis5 are asked to work with ISR.:

```
// header file declare
#include     "pci_8132.h"

PCI_INFO   info;

#define      axis1 1
#define      axis5 5
U16   int_flag=0, irq_axs;
U32   irq_sts;
```

```
    /*****************************************************************************/
    /*          MAIN   Program
*/
    /*****************************************************************************/
    void main( void )
    {
              U16       i, bn=0, status;
              _8132_Initial( &bn,   &info   );
     // Do System configuration for all I/O signals
    ……..
    //
    // Set Interrupt   factors for axis1, axis5
              set_int_factor(axis1, factor1);
              set_int_factor(axis5, factor2);
    // Enable Interrupt for both PCI-8132 cards
              for(i=0; i<bn; i++)
                   _8132_Set_INT_Enable(i, 1);
    // Main program for application
    ……..
    // End of Main Program
              for(i=0; i<bn; i++)
                   _8132_Close(i);                  // Close all IRQ resources
    }

    /*****************************************************************************/
    /*          ISR begin here
*/
    /*****************************************************************************/
    void interrupt _8132_isr0(void)
    {
              U16       int_axis;
              U16       irq_status;
    //
              disable();                // disable all interrupt
              _8132_Get_IRQ_Status(0, &irq_status);
              if(irq_status)            // Judge if INT for card 0?
               {
                 _8132_get_int_axis(&int_axis);
                 int_flag = 1;
                 irq_axs = int_axis;
                 _8132_get_int_status(int_axis, &irq_sts);
               }
              else
                  _chain_intr(pcinfo.old_isr[0]); // If not, chain to other INT
    //
```

```
        outportb(0x20, 0x20);       // End of INT
        outportb(0xA0, 0x20);
//-------------------------------------------------------------------------
        enable();                   // enable interrupt request

}

void interrupt _8132_isr1(void)
{
        U16     int_axis;
        U16     irq_status;
//
        disable();                  // disable all interrupt
        _8132_Get_IRQ_Status(1, &irq_status);
        if(irq_status)              // Judge if INT for card 1?
         {
           _8132_get_int_axis(&int_axis);
           int_flag = 1;
           irq_axs = int_axis;
           _8132_get_int_status(int_axis, &irq_sts);
         }
        else
            _chain_intr(pcinfo.old_isr[1]); // If not, chain to other INT
//
        outportb(0x20, 0x20);       // End of INT
        outportb(0xA0, 0x20);
//-------------------------------------------------------------------------
        enable();                   // enable interrupt request
}
```

So with the sample, user can get the interrupt signal about each axis in the motion control system.

5

Motion Creator

After installing all the hardware properly according to Chapter 2, 3, configuring cards and checkout are required before running. This chapter gives guidelines for establishing a control system and manually exercising the PCI-8132 cards to verify correct operation. Motion Creator provides a simple yet powerful means to setup, configure, test and debug motion control system that uses PCI-8132 cards.

Note that Motion Creator is available only for Windows 95/98 or Windows NT with the screen resolution higher than 800x600 environment and can not run on DOS.

5.1 Main Menu

Main Menu will appear when executing Motion Creator. Figure 5.1 shows the Main Menu.

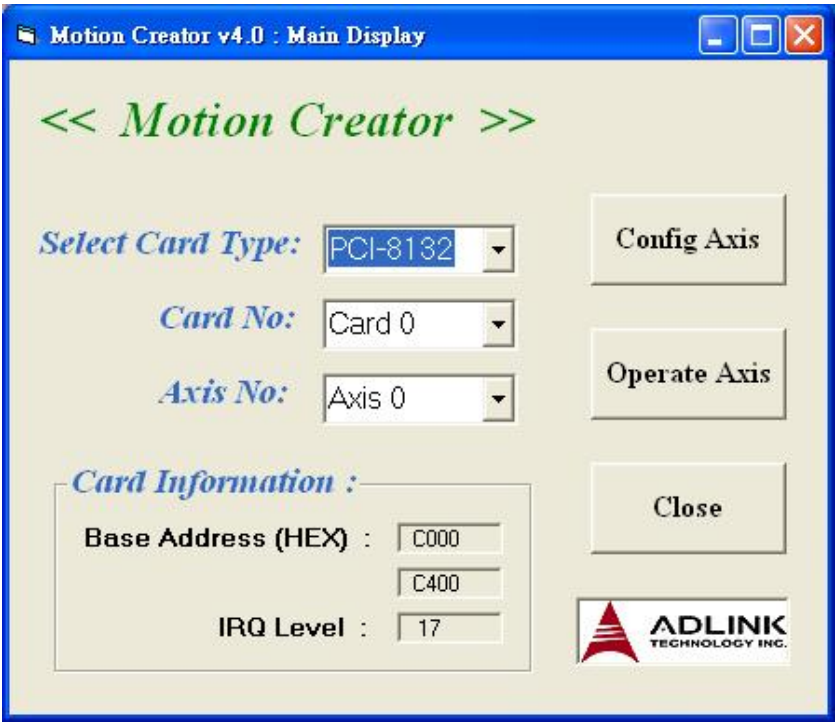
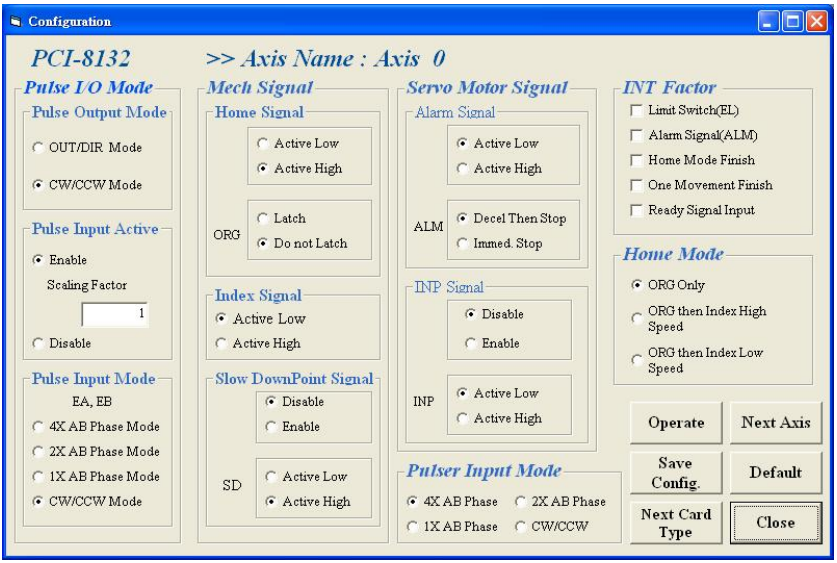


Figure 5.1 Main Menu of Motion Creator

From main menu window all PCI-8132 cards and their axes and the corresponding status can be viewed. First of all, check if all the PCI-8132 cards which are plugged in the PCI-Bus can be viewed on “Select Card” column. Next select the card and axis you want to configure and operate. Since there are totally four axes on a card, the axis number of first axis on n-the card will be numbered as 4*(n-1). Base address and IRQ level of the card are also shown on this window.

5.2 Axis Configuration Window



Press the “Config Axis” button on the Main Menu will enter the Axis Configuration window. Figure 5.2 shows the window.

Figure 5.2 Axis Configuration Window

the Axis Configuration window includes the following setting items which cover most I/O signals of PCI-8132 cards and part of the interrupt factors.

- **Pulse I/O Mode:**

Related functions:

- set_pls_outmode() for “Pulse Output Mode” property.
- set_cnt_src() for “Pulse Input Active” property.
- set_pls_iptmode() for “Pulse Input Mode” property.

- **Mechanical Signal:**

  Related functions:

  - set_home_config() for “Home Signal” and “Index Signal” property.
  - set_sd_logic() for “Slow Down Point Signal” property.

- **Servo Motor Signal:**

  Related functions:

  - set_alm_logic() for “Alarm Signal” property.
  - set_inp_logic() for “INP” property.

- **Manual Pulser Input Mode:**

  Related functions:

  - set_manu_iptmode() for “Manual Pulser Input Mode” property.

- **Interrupt Factor:**

  Related functions:

  - set_int_factor() for “INT Factor” property.

- **Home Mode:**

  Related functions:

  - set_home_config() for “Home Mode” property.

The details of each section are shown at its related functions.

After selecting all the items you want to configure, user can choose to push the “Save Configurations “ button on the right bottom side. If you push this button, all the configurations you select for system integration will be saved to a file called “8132.cfg”. This file is very helpful when user is developing their own application programs. The following example illustrate how to make use of this function. This example program is shown in C language form.

```
Main()
{
    _8132_initial();          // Initialize the PCI-8132 cards
    _8132_Set_Config();       // Configure PCI-8132 cards according
 :                                                          // to
8132.cfg
 :
}
```

Where *_8132_initial()* and *_8132_Set_Config()* can be called from the function library we provide. *_8132_initial()* should be the first function called within main{} function. It will check all the PCI-8132 existed and give the card a base address and IRQ level. *_8132_Set_Config()* will configure the PCI-8132 cards according to "8132.cfg". That is, the contents of Axis Configuration Window can be transferred to the application program by this function called.

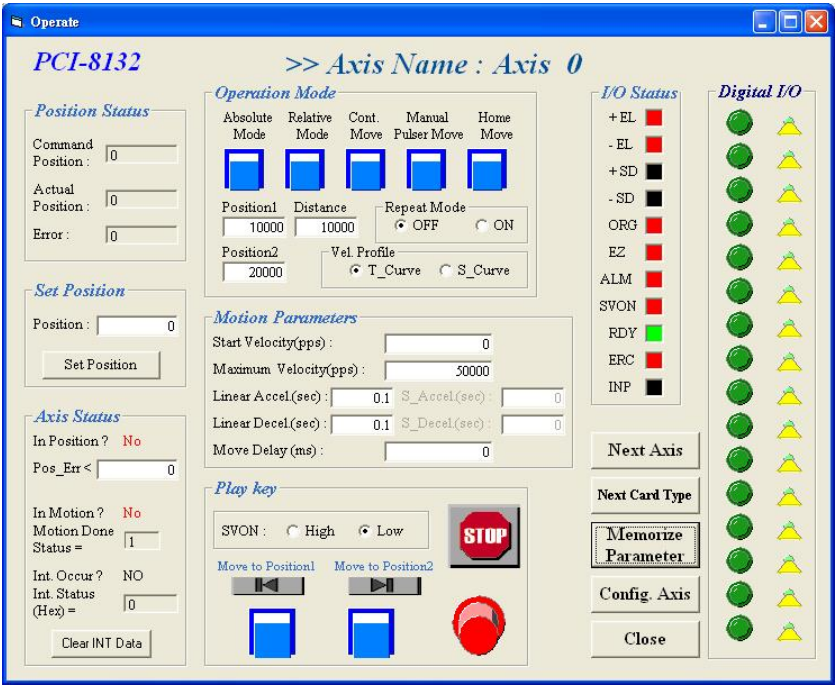


Figure 5.3 Axis Operation window

5.3 Axis Operation Windows

Press the “Operate Axis” button on the Main Menu or Axis Configuration Menu will enter the Axis Configuration window. Figure 5.3 shows the window. User can use this window to command motion, monitor all the I/O status for the selected axis. This window includes the following displays and controls:

- Motion Status Display,
- Axis Status Display
- I/O Status Display
- Set Position Control
- Operation Mode Control
- Motion Parameter Control
- Play Key Control
- Velocity Profile Selection
- Repeat Mode

5.3.1 Motion Status Display

The Motion Status display provides a real-time display of the axis’s position in the Command, Actual, Error fields. Motion Creator automatically updates these command, actual and error displays whenever any of the values change.

When Pulse Input Active property is Axis Configuration Window is set to Enable, the Actual Position read will be from the external encoder inputs(EA, EB). Else, it will display the command pulse output when set to Disable.

5.3.2 Axis Status Display

The Axis Status display provides a real-time display of the axis’s status.

It displays the status(Yes(for logical True) or No(for logical False)) for In Position or In Motion or displays there is Interrupt Events Occurs. When In motion, you can check the motion done status in the next column. In Position range can be specified in the Pos_Err column.

5.3.3 I/O Status Display

Use I/O Status display to monitor the all the I/O status of PCI-8132. The Green Light represents ON status, Red Light represents OFF status and BLACK LIGHT represents that I/O function is disabled. The ON/OFF status is read based on the setting logic in Axis Configuration window.

5.3.4 Set Position Control

Use the Set Position Control to arbitrarily change the actual position of axis.

Write the position wanting to specify into the column and click the “Set Position” button will set the actual position to the specified position.

5.3.5 Operation Mode Control

There are four Operation Modes mentioned in Chapter 4 can be tested in the Axis Operation window. They are “Continuous Move Mode”, “Preset Mode Operation”, “Home Mode Operation”, “Manual Mode Operation”.

- **Continuous Move Mode:**

Press “Continuous Move” button will enable Continuous Velocity motion as specified by values entered in “Start Velocity” and “Maximum Velocity” 2 fields of Motion Parameters Control. The steady state moving velocity will be as specified by “Maximum Velocity”. Press → to move forward or ← to move backward. Press “STOP” to stop moving.

- **Preset Mode:**

Press “Absolute Mode” to enable absolute motion as specified by values entered in “Position 1” and “Position 2” 2 fields. When selected, “Distance” field for “Relative Mode” is disabled. Press → to move to Position 2 or ← to move to Position 1. Press “STOP” to stop motion.

Also, user can specify repetitive motion in “Absolute Mode” by setting “Repeat Mode” to “ON” state. When “Repeat Mode” goes “ON” and either → or ← is pressed., axis starts repetitive motion between Position 1 and Position 2 until “Repeat Mode” goes “OFF” as “STOP” are clicked.

Press “Relative Mode” to enable relative motion as specified by values entered in “Distance” fields. When selected, “Position 1” and “Position 2” fields for “Absolute Mode” is disabled. Press → to move forward to a distance relative to present position as specified by “Distance” or ← to move backward.

Note that both “Absolute Mode” and “Relative Mode” are operated under a trapezoidal velocity profile as specified by Motion Parameters Control.

♦ **Home Return Mode:**

Press “Home Move” button will enable Home Return motion. The home returning velocity is specified by settings in Motion Parameters Control. The arriving condition for Home Return Mode is specified in Axis Configuration Window. Press → to begin returning home function. Press “STOP” to stop moving.

♦ **Manual Pulser Mode:**

Press “Manual Pulser Move” button will enable motion controlled by hand wheel pulser. Using this function, user can manually operate the axis thus verify operation. The maximum moving velocity is limited as specified by “Maximum Velocity”. Press “STOP” to end this mode.
Do remember to press “STOP” to end operation under this mode. Otherwise, operations under other modes will be inhibited.

5.3.6 Motion Parameters Control

Use the Motion Parameters with the Operation Mode Control to command motion.

- Starting Velocity: Specify the starting moving speed in pulses per second.
- Maximum Velocity: Specify the maximum moving speed in pulses per second.
- Acceleration: Specify the acceleration in pulses per second square.
- Move delay: Specify time in mini seconds between movement.
- S curve Acc/dec Time: Specify time in mini second for S_curve Movement.

5.3.7 Play Key Control

Use buttons in Play Key Control to begin or end operation.



: click button under this symbol to begin moving to Positions 2 in Absolute Mode or moving forward in other modes.



: click button under this symbol to begin moving to Positions 1 in Absolute Mode or moving backward in other modes.



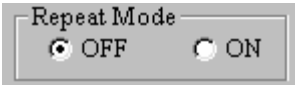
: click button under this symbol to stop motion under any mode. Note that this button is always in latch mode. Click again to release “STOP” function.

5.3.8 Velocity Profile Selection



: Click T_Curve or S_curve to select preset movement velocity profile. The relative parameter settings are in Motion Parameter Frame.

5.3.9 Repeat Mode



: Repeat mode is only for absolute and relative mode. After choosing a operation mode and click repeat mode on, you can press play key to make axis run between position 1 and 2 (in absolute mode) or run between a range (relative mode). It is useful on demonstrations. Use Stop button to stop this operation.

6

Function Library

This chapter describes the supporting software for PCI-8132 cards. User can use these functions to develop application program in C or Visual Basic or C++ language.

6.1 List of Functions

Initialization	***Section 6.3***
_8132_Initial(card_no);	Software initialization
_8132_InitialA(TotalCard)	Software initialization
_8132_Close(card_no);	Software Close
_8132_Set_Config(void);	Configure PCI-8132 according to Motion Creator

Pulse Input/Output Configuration	***Section 6.4***
_8132_set_pls_outmode(axis, pls_outmode);	Set pulse command output mode
_8132_set_pls_iptmode(axis, pls_iptmode);	Set encoder input mode
_8132_set_cnt_src(axis, cnt_src);	Set counter input source

Continuously Motion Mode	***Section 6.5***
_8132_v_move(axis, svel, mvel, Tacc);	Accelerate an axis to a constant velocity with trapezoidal profile
_8132_sv_move(axis, svel, mvel, Tlacc, Tsacc);	Accelerate an axis to a constant velocity with S-curve profile
_8132_v_change(axis, mvel, Tacc);	Change speed on the fly
_8132_v_stop(axis, Tdec);	Decelerate to stop
_8132_fix_max_speed(axis,max_speed);	Fix max speed for v_change

Trapezoidal Motion Mode	***Section6.6***
_8132_a_move(axis, pos, svel, mvel, Tacc);	Perform an absolute trapezoidal

	profile move
_8132_start_a_move(axis, pos, svel, mvel, Tacc);	Begin an absolute trapezidal profile move
_8132_r_move(axis, dist, svel, mvel, Tacc);	Perform a relative trapezoidal profile move
_8132_start_r_move(axis, dist, svel, mvel, Tacc);	Begin a relative trapezoidal profile move
_8132_t_move(axis, dist, svel, mvel, Tacc, Tdec);	Perform a relative non-symmetrical trapezoidal profile move
_8132_start_t_move(axis, dist, svel, mvel, Tacc, Tdec);	Begin a relative non-symmetrical trapezidal profile move
_8132_start_ta_move(axis, pos, svel, mvel, Tacc, Tdec);	Begin an absolute non-symmetrical trapezidal profile move
_8132_ta_move(axis, pos, svel, mvel, Tacc, Tdec);	Perform an absolute non-symmetrical trapezoidal profile move
_8132_wait_for_done(axis);	Wait for an axis to finish

S-Curve Profile Motion	***Section 6.7***
_8132_s_move(axis, pos, svel, mvel, Tlacc, Tsacc);	Perform an absolute S-curve profile move
_8132_start_s_move(axis, pos, svel, mvel, Tlacc, Tsacc);	Begin an absolute S-curve profile move
_8132_rs_move(axis, dist, svel, mvel, Tlacc, Tsacc);	Perform a relative S-curve profile move
_8132_start_rs_move(axis, dist, svel, mvel, Tlacc, Tsacc);	Begin a relative S-curve profile move
_8132_tas_move(axis, pos, svel, mvel, Tlacc, Tsacc, Tldec, Tsdec);	Perform an absolute non-symmetrical S-curve profile move
_8132_start_tas_move(axis, pos, svel, mvel, Tlacc, Tsacc, Tldec, Tsdec);	Begin an absolute non-symmetrical S-curve profile move

Multiple Axes Point to Point Motion	***Section 6.8***
_8132_start_move_all(n_axes, *axes, *pos, *svel, *mvel, *Tacc);	Begin a multi-axis trapezodial profile move
_8132_move_all(n_axes, *axes, *pos, *svel, *mvel, *Tacc);	Perform a multi-axis trapezodial profile move
_8132_wait_for_all(n_axes, *axes);	Wait for all axes to finish

Linear Interpolated Motion	***Section 6.9***
_8132_move_xy(cardNo, x, y);	2-axis linear interpolated move for X & Y
_8132_start_move_xy(cardNo, x, y)	2-axis linear interpolated move for X & Y
_*8132*_recover_xy(axisno);	Return to single axis mode

Interpolation Parameters Configuring	***Section 6.10***
_8132_map_axes(n_axes, *map_array);	Maps coordinated motion axes x, y, z….
_8132_set_move_speed(svel, mvel);	Set the vector velocity
_8132_set_move_accel(Tacc);	Set the vector acceleration time
_8132_set_move_ratio(axis, ratio);	Set the axis resolution ratios

Home Return Mode	***Section 6.11***
_8132_set_home_config(axis, mode, org_logic, org_latch, index_logic);	Set or get the home/index logic configuration
_8132_home_move(axis, svel, mvel, accel);	Begin a home return action

Manual Pulser Motion	***Section 6.12***
_8132_set_manu_iptmode(axis, ipt_mode, op_mode);	Set pulser input mode and operation mode
_8132_manu_move(axis, mvel);	Begin a manual pulser movement
_8132_set_manu_axis(cardno, manu_axis);	Select manual pulser axis
_8132_set_step_unit(axisno, unit)	Set manual pulser ratio

Motion Status	***Section 6.13***
_8132_motion_done(axis);	Returns TRUE if motion done

Servo Drive Interface	***Section 6.14***
_8132_set_alm_logic(axis, alm_logic, alm_mode);	Set alarm logic and alarm mode
_8132_set_inp_logic(axis, inp_logic, inp_enable);	Set In-Position logic and enable/disable
_8132_set_sd_logic(axis, sd_logic, sd_latch, sd_enable);	Set slow down point logic and enable/disable
_8132_set_erc_enable(axis, erc_enable)	Set the ERC output enable/disable

I/O Control and Monitoring	***Section 6.15***
_8132_Set_SVON(axis, on_off);	Set the state of general purpose output bit
_8132_get_io_status(axis, *io_status);	Get all the I/O staus of PCI-8132

Position Control	***Section 6.16***
_8132_set/get_position(axis, pos);	Set or get current actual position
_8132_set/get_command(axis, pos);	Set or get current command position

Interrupt Control	***Section 6.17***
_8132_Set_INT_ENABLE (cardNo, intFlag)	Set Interrupt enable (for DOS

	only)
_8132_INT_Enable(card_number, phEvent)	Set Interrupt enable
_8132_INT_Disable(card_number)	Set Interrupt disable
_8132_Set_INT_Control(cardNo, int1Flag, int2Flag);	Enable/Disable IRQ channel
_8132_set_int_factor(axis, int_factor);	Set Interrupt generationg factors
_8132_get_int_axis(*int_axis);	Get the axis which generates interrupt (DOS)
_8132_get_int_status(axis, *int_status);	Get the interrupting status of axis

Digital I/O Control	***Section 6.18***
_8132_DO(axis, DoData);	Output digital channel
_8132_DI(axis, *DiData);	Input digital channel

Position Compare Control	***Section 6.19***
_8132_Get_CompCnt	Get counter value from comparator
_8132_Set_CompCnt	Set counter value in comparator
_8132_Set_CompMode	Set compare mode
_8132_Set_CompData	Set comparator value
_8132_Get_CompData	Get current comparator value
_8132_Set_CompInt	Enable comparator Interrupt
_8132_Set_CompHome	Set comparator origin
_8132_Get_CompSts	Get comparator status
_8132_Build_Comp_Table	Build compare table
_8132_Set_Comp_Table	Enable/Disable compare table
_8132_Build_Comp_Function	Build a linear trigger table by a function

6.2 C/C++ Programming Library

This section gives the details of all the functions. The function prototypes and some common data types are decelerated in **PCI-8132.H**. These data types are used by PCI-8132 library. We suggest you to use these data types in your application programs. The following table shows the data type names and their range.

Type Name	Description	Range
U8	8-bit ASCII character	0 to 255
I16	16-bit signed integer	-32768 to 32767
U16	16-bit unsigned integer	0 to 65535
I32	32-bit signed long integer	-2147483648 to 2147483647
U32	32-bit unsigned long integer	0 to 4294967295
F32	32-bit single-precision floating-point	-3.402823E38 to 3.402823E38
F64	64-bit double-precision floating-point	-1.797683134862315E308 to 1.797683134862315E309
Boolean	Boolean logic value	TRUE, FALSE

The functions of PCI-8132's software drivers use full-names to represent the functions' real meaning. The naming convention rules are :

In DOS Environment :

_{hardware_model}_{action_name}. e.g. **_8132_Initial()**.

In order to recognize the difference between C and VB function, A capital "B" is put on the head of each function name of the Visual Basic. e.g. **B_8132_Initial()**.

6.3 Initialization

@ *Name*

_8132_Initial – Software Initialization for PCI-8132
_8132_InitialA – Software Initialization for PCI-8132
_8132_Close – Software release resources of PCI-8132
_8132_Set_Config – Configure PCI-8132 according to Motion Creator
_8132_Get_IRQ_Channel – Get the PCI-8132 card's IRQ number
_8132_Get_ Base_Addr – Get the PCI-8132 card's base address

@ *Description*

_8132_Initial :

This function is used to initialize PCI-8132 card. Every PCI-8132 card has to be initialized by this function before calling other functions.

_8132_InitialA :

This function is used to initialize all PCI-8132 cards in the host system. Every PCI-8132 card has to be initialized by this function before calling other functions.

_8132_Close :

This function is used to close PCI-8132 card and release the PCI-8132 related resources, which should be called at the end of an application.

_8132_Set_Config :

This function is used to configure PCI-8132 card. All the I/O configurations and some operating modes appeared on "Axis Configuration Window" of Motion Creator will be set to PCI-8132. Click "Save Configuration" button on the "Axis Configuration Window" if you want to use this function in the application program. Click "Save Configuration" button will save all the configurations to a file call "*8132.cfg*". This file will appear in the "WINDOWS\SYSTEM\" directory.

_8132_Get_IRQ_Channel :

This function is used to get the PCI-8132 card's IRQ number. (This function just suport Window 95 and Window NT platform only).

_8132_Get_Base_Addr :

This function is used to get the PCI-8132 card's base address. (This function just suport Window 95 and Window NT platform only).

@ *Syntax*

C/C++ (DOS)

```
U16 _8132_Initial (U16 *existCards, PCI_INFO *info)
U16 _8132_Close(U16 cardNo)
U16 _8132_Set_Config(char* filename)
```

C/C++ (Windows 95/NT)

```
U16 _8132_Initial(U16 *existCards, PCI_INFO *pciInfo) (Windows 95 Only)
U16 _8132_Initial(U16 cardNo)(Windows NT Only)
```

U16 _8132_InitialA(I16 *TotalCard)
U16 _8132_Close(U16 cardNo)(Windows NT Only)
U16 _8132_Set_Config(char *fileName)
void _8132_Get_IRQ_Channel(U16 cardNo, U16 *irq_no)
void _8132_Get_Base_Addr(U16 cardNo, U16 *base_addr)

Visual Basic (Windows 95/NT)

B_8132_Initial (existCards As Integer, pciInfo As PCI_INFO) As Integer(Windows 95 Only)
B_8132_Initial (ByVal cardNo As Long) As Integer (Windows NT Only)
B_8132_InitialA(TotalCard as Integer) As Interger
B_8132_Close (ByVal cardNo As Long) As Integer (Windows NT Only)
B_8132_Set_Config (ByVal fileName As String) As Integer
B_8132_Get_IRQ_Channel (ByVal cardno As Integer, irq_no As Integer)
B_8132_Get_Base_Addr (ByVal cardno As Integer, base_addr As Integer)

@ *Argument*

existCards: numbers of existing PCI-8132 cards
TotalCard: numbers of existing PCI-8132 cards
info: relative information of the PCI-8132 cards
cardNo: The PCI-8132 card index number.

@ *Return Code*

ERR_NoError
ERR_BoardNoInit
ERR_PCIBiosNotExist

6.4 Pulse Input / Output Configuration

@ Name

_8132_set_pls_outmode – Set the configuration for pulse command output.

_8132_set_pls_iptmode – Set the configuration for feedback pulse input.

_8132_set_cnt_src – Enable/Disable the external feedback pulse input

@ Description

_8132_set_pls_outmode:

Configure the output modes of command pulse. There are two modes for command pulse output.

_8132_set_pls_iptmode:

Configure the input modes of external feedback pulse. There are four types for feedback pulse input. Note that this function makes sense only when **cnt_src** parameter in **set_cnt_src()** function is enabled.

_8132_set_cnt_src:

If external encoder feedback is available in the system, set the **cnt_src** parameter in this function to *Enabled* state. Then internal 28-bit up/down counter will count according configuration of **set_pls_iptmode()** function.

Or the counter will count the command pulse output.

@ Syntax

C/C++ (DOS, Windows 95/NT)

U16 _8132_set_pls_outmode(I16 axis, I16 pls_outmode)
U16 _8132_set_pls_iptmode(I16 axis, I16 pls_iptmode)
U16 _8132_set_cnt_src(I16 axis, I16 cnt_src)

Visual Basic (Windows 95/NT)

B_8132_set_pls_outmode (ByVal axis As Long, ByVal pls_outmode As Long) As Integer
B_8132_set_pls_iptmode (ByVal axis As Long, ByVal pls_iptmode As Long) As Integer
B_8132_set_cnt_src (ByVal axis As Long, ByVal cnt_src As Long) As Integer

@ Argument

axis: axis number designated to configure pulse Input/Output.
pls_outmode: setting of command pulse output mode for OUT and DIR pins.
pls_outmode=0, OUT/DIR type pulse output.
pls_outmode=1, CW/CCW type pulse output.
pls_iptmode: setting of encoder feedback pulse input mode for EA and EB pins.

pls_iptmode=0, 1X AB phase type pulse input.
pls_iptmode=1, 2X AB phase type pulse input.
pls_iptmode=2, 4X AB phase type pulse input.
pls_iptmode=3, CW/CCW type pulse input.

cnt_src: Counter source

cnt_src=0, counter source from command pulse
cnt_src=1, counter source from external input EA, EB

@ Return Code

ERR_NoError

6.5 Continuously Motion Move

@ Name

_8132_v_move – Accelerate an axis to a constant velocity with trapezoidal profile

_8132_sv_move – Accelerate an axis to a constant velocity with S-curve profile

_8132_v_change – Change speed on the fly

_8132_v_stop – Decelerate to stop

_8132_fix_max_speed – Set max speed when using v_change() function

@ Description

_8132_v_move:

This function is used to accelerate an axis to the specified constant velocity. The axis will continue to travel at a constant velocity until the velocity is changed or the axis is commanded to stop. The direction is determined by the sign of velocity parameter.

_8132_sv_move:

This function is similar to v_stop() but accelerating with S-curve.

_8132_v_change:

You can change the velocity profile of command pulse ouput during operation by this function. This function changes the maximum velocity setting during operation. However, if you operate under “Preset Mode” (like start_a_move(),…), you are not allowed to change the acceleration parameter during operation because the deceleration point is pre-determined. But changing the acceleration parameter when operating under “Constant Velocity Mode” is valid.

_8132_fix_max_speed:

In order to calculate better performance when using v_change() function, user must set this function before any PTP function

_8132_v_stop:

This function is used to decelerate an axis to stop. This function is also useful when ***preset move***(both trapezoidal and S-curve motion),

manual move or ***home return*** function is performed.

@ Syntax

C/C++ (DOS, Windows 95/NT)

U16 _8132_v_move(I16 axis, F64 str_vel, F64 max_vel, F64 Tacc)
U16 _8132_sv_move(I16 axis, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc)
U16 _8132_v_change(I16 axis, F64 max_vel, F64 Tacc)
U16 _8132_fix_max_speed(I16 axis, F64 max_vel)
U16 _8132_v_stop(I16 axis, F64 Tdec)

Visual Basic (Windows 95/NT)

B_8132_v_move (ByVal axis As Integer, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tacc As Double) As Integer
B_8132_sv_move(I16 axis, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc) As Integer
B_8132_v_change(I16 axis, F64 max_vel, F64 Tacc) As Integer
B_8132_fix_max_speed (ByVal axis As Integer, ByVal max_speed As Double) As Integer
B_8132_v_stop (ByVal axis As Integer, ByVal Tacc As Double) As Integer

@ Argument

axis: axis number designated to move or stop.
str_vel: starting velocity in unit of pulse per second
max_vel: maximum velocity in unit of pulse per second
max_speed: maximum velocity during a v_change() function
Tacc: specified acceleration time in unit of second
Tdec: specified deceleration time in unit of second

@ Return Code

ERR_NoError

6.6 Trapezoidal Motion Mode

@ Name

_8132_start_a_move– Begin an absolute trapezoidal profile motion
_8132_start_r_move– Begin a relative trapezoidal profile motion
_8132_start_t_move– Begin a non-symmetrical relative trapezoidal profile motion
_8132_start_ta_move– Begin a non-symmetrical absolute trapezoidal profile motion
_8132_a_move– Begin an absolute trapezoidal profile motion and wait for completion
_8132_r_move– Begin a relative trapezoidal profile motion and wait for completion
_8132_t_move– Begin a non-symmetrical relative trapezoidal profile motion and wait for completion

_8132_ta_move– Begin a non-symmetrical absolute trapezoidal profile motion and wait for completion

@ *Description*

_8132_start_a_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position, immediately returning control to the program. The acceleration rate is equal to the deceleration rate. **_8132_a_move()** starts an absolute coordinate move and waits for completion.

_8132_start_r_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the relative distance, immediately returning control to the program. The acceleration rate is equal to the deceleration rate. **_8132_r_move()** starts a relative move and waits for completion.

_8132_start_ta_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position, immediately returning control to the program.. **_8132_ta_move()** starts an absolute coordinate move and waits for completion.

_8132_start_t_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the relative distance, immediately returning control to the program.. **_8132_t_move()** starts a relative coordinate move and waits for completion.

The moving direction is determined by the sign of ***pos*** or ***dist*** parameter.If the moving distance is too short to reach the specified velocity, the controller will accelerate for the first half of the distance and decelerate for the second half (triangular profile). **wait_for_done()** waits for the motion to complete.

@ *Syntax*

C/C++ (DOS, Windows 95/NT)

U16 _8132_start_a_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tacc)
U16 _8132_a_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tacc)
U16 _8132_start_r_move(I16 axis, F64 distance, F64 str_vel, F64 max_vel, F64 Tacc)
U16 _8132_r_move(I16 axis, F64 distance, F64 str_vel, F64 max_vel, F64 Tacc)

U16 _8132_start_t_move(I16 axis, F64 dist, F64 str_vel, F64 max_vel, F64 Tacc, F64 Tdec)
U16 _8132_t_move(I16 axis, F64 dist, F64 str_vel, F64 max_vel, F64 Tacc, F64 Tdec)
U16 _8132_start_ta_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tacc, F64 Tdec)
U16 _8132_ta_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tacc, F64 Tdec)
U16 _8132_wait_for_done(I16 axis)

Visual Basic (Windows 95/NT)

B_8132_start_a_move (ByVal axis As Integer, ByVal pos As Double, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tacc I As Double) As Integer
B_8132_a_move (ByVal axis As Integer, ByVal pos As Double, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tacc As Double) As Integer
B_8132_start_r_move (ByVal axis As Integer, ByVal distance As Double, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tacc As Double) As Integer
B_8132_r_move (ByVal axis As Integer, ByVal distance As Double, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tacc As Double) As Integer
B_8132_start_t_move (ByVal axis As Integer, ByVal distance As Double, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer
B_8132_t_move (ByVal axis As Integer, ByVal distance As Double, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer
B_8132_start_ta_move(ByVal axis As Integer, ByVal pos As Double , ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Interger
B_8132_ta_move(ByVal axis As Integer, ByVal pos As Double , ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer
B_8132_wait_for_done(ByVal axis As Integer) As Integer

@ *Argument*

axis: axis number designated to move.
pos: specified absolute position to move
distance or dist: specified relative distance to move
str_vel: starting velocity of a velocity profile in unit of pulse per second
max_vel: starting velocity of a velocity profile in unit of pulse per second
Tacc: specified acceleration time in unit of second
Tdec: specified deceleration time in unit of second

@ *Return Code*

ERR_NoError
ERR_MoveError

6.7 S-Curve Profile Motion

@ Name

_8132_start_s_move– Begin a S-Curve profile motion

_8132_s_move– Begin a S-Curve profile motion and wait for completion

_8132_start_rs_move– Begin a relative S-Curve profile motion

_8132_rs_move– Begin a relative S-Curve profile motion and wait for completion

_8132_start_tas_move– Begin a non-symmetrical absolute S-curve profile motion

_8132_tas_move– Begin a non-symmetrical absolute S-curve profile motion and wait for completion

@ Description

_8132_start_s_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position, immediately returning control to the program. The acceleration rate is equal to the deceleration rate. **_8132_s_move()** starts an absolute coordinate move and waits for completion.

_8132_start_rs_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the relative distance, immediately returning control to the program. The acceleration rate is equal to the deceleration rate. **_8132_rs_move()** starts a relative move and waits for completion.

_8132_start_tas_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position, immediately returning control to the program.. **_8132_tas_move()** starts an absolute coordinate move and waits for completion.

@ Syntax

C/C++ (DOS, Windows 95/NT)

U16 _8132_start_s_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc)

U16 _8132_s_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc)

U16 _8132_start_rs_move(I16 axis, F64 distance, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc)

U16 _8132_rs_move(I16 axis, F64 distance, F64 str_vel, F64 max_vel, F64

Tlacc, F64 Tsacc)
U16 _8132_start_tas_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc, F64 Tldec, F64 Tsdec)
U16 _8132_tas_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc, F64 Tldec, F64 Tsdec)

Visual Basic (Windows 95/NT)

B_8132_start_s_move(ByVal axis As Integer, ByVal pos As Double, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tlacc As Double, ByVal Tsacc As Double) As Integer
B_8132_s_move(ByVal axis As Integer, ByVal pos As Double, ByVal str_vel As Double, ByVal max_vel As Double ByVal Tlacc As Double, ByVal Tsacc As Double) As Integer
B_8132_start_rs_move(ByVal axis As Integer, ByVal distance As Double, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tlacc As Double, ByVal Tsacc As Double) As Integer
B_8132_rs_move(ByVal axis As Integer, ByVal distance As Double, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tlacc As Double, ByVal Tsacc As Double) As Integer
B_8132_start_tas_move(ByVal axis As Integer, ByVal pos As Double, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tlacc As Double, ByVal Tsacc As Double, ByVal Tldec As Double, ByVal Tsdec As Double) As Integer
B_8132_tas_move(ByVal axis As Integer, ByVal pos As Double ByVal str_vel As Double, ByVal max_vel As Double ByVal Tlacc As Double, ByVal Tsacc As Double, ByVal Tldec As Double, ByVal Tsdec As Double) As Integer

@ *Argument*

axis: axis number designated to move.
pos: specified absolute position to move
distance or dist: specified relative distance to move
str_vel: starting velocity of a velocity profile in unit of pulse per second
max_vel: starting velocity of a velocity profile in unit of pulse per second
Tlacc: specified linear acceleration time in unit of second
Tsacc: specified S-curve acceleration time in unit of second
Tldec: specified linear deceleration time in unit of second
Tsdec: specified S-curve deceleration time in unit of second

@ *Return Code*

ERR_NoError
ERR_MoveError

6.8 Multiple Axes Point to Point Motion

@ Name

_8132_start_move_all– Begin a multi-axis trapezoidal profile motion

_8132_move_all–Begin a multi-axis trapezoidal profile motion and wait for completion

_8132_wait_for_all–Wait for all axes to finish

@ Description

_8132_start_move_all() :

This function causes the specified axes to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position, immediately returning control to the program. The move axes are specified by axes and the number of axes are defined by ***n_axes***. The acceleration rate of all axes is equal to the deceleration rate. **_8132_move_all()** starts the motion and waits for completion. Both functions guarantee that motion begins on all axes at the same sample time. **Note** that it is necessary to make connections according to Section 3.12 on CN3 if these two functions are needed.

_8132_wait_for_done() waits for the motion to complete for all of the specified axes.

The following code demos how to utilize these functions. This code moves axis 0 and axis 4 to position 8000.0 and 120000.0 respectively. If we choose velocities and acelerations that are propotional to the ratio of distances, then the axes will arrive at their endpoints at the same time (simultaneous motion).

```
#include        “pci_8132.h”

int main()
{
        I16     axes[2] = {0, 4};
        F64
         positions[2] = {8000.0, 12000.0},
         str_vel[2]={0.0, 0.0},
         max_vel[2]={4000.0, 6000.0},
         Tacc[2]]={0.04, 0.06};

_8132_move_all(2, axes, positions, str_vel, max_vel, Tacc);

return   ErrNoError;
}
```

@ *Syntax*

C/C++ (DOS, Windows 95/NT)

U16 _8132_start_move_all(I16 len, I16 *axes, F64 *pos, F64 *str_vel, F64 *max_vel, F64 *Tacc)

U16 _8132_move_all(I16 len, I16 *axes, F64 *pos, F64 *str_vel, F64 *max_vel, F64 *Tacc)

U16 _8132_wait_for_all(I16 len, I16 *axes)

Visual Basic (Windows 95/NT)

B_8132_start_move_all(ByVal len As Integer, ByRef axis As Integer , ByRef pos As Double, ByRef str_vel As Double, ByRef max_vel As Double, ByRef Tacc As Double) As Integer

B_8132_move_all(ByVal len As Integer, ByRef axis As Integer, ByRef pos As Double, ByRef str_vel As Double, ByRef max_vel As Double, ByRef Tacc As Double) As Integer

B_8132_wait_for_all(ByVal n_axes As Integer, ByRef axis As Integer) As Integer

@ *Argument*

n_axes: number of axes for simultaneous motion
***axes**: specified axes number array designated to move.
***pos**: specified position array in unit of pulse
***str_vel**: starting velocity array in unit of pulse per second
***max_vel**: maximum velocity array in unit of pulse per second
***Tacc**: acceleration time array in unit of second

@ *Return Code*

ERR_NoError
ERR_MoveError

6.9 Linear and Circular Interpolated Motion

@ Name

_8132_start_move_xy – Perform a 2-axes linear interpolated motion between X & Y without waiting

_8132_move_xy – Perform a 2-axes linear interpolated motion between X & Y and wait for completion

_8132_recover_xy – return single axis motion mode

@ Description

_8132_move_xy, _8132_start_move_xy:

These two functions cause a linear interpolation motion between two axes and wait for completion. The moving speed should be set before performing these functions. Relations of speed between two axes are given in Chapter 4.1.4.

_8132_recover_xy:

After using *_start_move_xy, use must use this function for next single PTP axis motion*

@ Syntax

C/C++ (DOS, Windows 95/NT)

```
U16 _8132_move_xy(I16 cardNo, F64 x, F64 y)
U16 _8132_start_move_xy(I16 cardNo, F64 x, F64 y)
U16 _8132_recover_xy(int cardNo)
```

Visual Basic (Windows 95/NT)

```
B_8132_move_xy (ByVal cardno As Long, ByVal x As Double, ByVal y As
        Double) As Integer
B_8132_start_move_xy (ByVal cardno As Long, ByVal x As Double, ByVal y
        As Double) As Integer
B_8132_recover_xy (ByVal cardno As Long) As Integer
```

@ Argument

cardNo: card number designated to perform interpolating function.
x, y: absolute target position of linear interpolation motion

@ Return Code

ERR_NoError

6.10 Interpolation Parameters Configuring

@ Name

_8132_map_axes – Configure the axis map for coordinated motion

_8132_set_move_speed – Set the vector velocity

_8132_set_move_accel – Set the vector linear acceleration time
_8132_set_move_ratios – Set the axis resolution ratios

@ *Description*

map_axes:

This function initializes a group of axes for coordinated motion. **map_axes()** must be called before any coordinated motion function is used. For PCI-8132, coordinated motion is made only between two axes. For example, if the z and u coordinates correspond to axes 2 and 3, the following code would be used to define the coordinate system:

```
int ax[2] = {2, 3};
map_axes(2, ax);
set_move_speed(10000.0);    // Set vector velocity = 10000pps
set_move_accel(0.1);    // Set vector accel. time = 0.1 sec
```

set_move_speed, set_move_accel:

The vector velocity and vector acceleration can be specified for coordinated motion by this three functions. Codes at last samples demonstrates how to utilize this three functions associated with **map_axes()**.

set_move_ratio:

This function configures scale factors for the specified axis. Usually, the axes only need scale factors if their mechanical resolutions are different. For example, if the resolution of feedback sensors is two times resolution of command pulse, then ***ratio = 2***.

@ *Syntax*

C/C++ (DOS, Windows 95/NT)

```
U16 _8132_map_axes(U16 n_axes, U16 *map_array)
U16 _8132_set_move_speed(F64 str_vel, F64 max_vel)
U16 _8132_set_move_accel(F64 Tacc)
U16 _8132_set_move_ratio(U16 axis, F64 ratio)
```

Visual Basic (Windows 95/NT)

```
B_8132_map_axes (ByVal n_axes As Integer, map_array As Integer) As
        Integer
B_8132_set_move_speed (ByVal str_vel As Double, ByVal max_vel As
        Double) As Integer
B_8132_set_move_accel (ByVal accel As Double) As Integer
B_8132_set_move_ratio (ByVal axis As Integer, ByVal ratio As Double) As
        Integer
```

@ *Argument*

axis: axis number designated to configure
n_axes: number of axes for coordinated motion

***map_array**: specified axes number array designated to move.
str_vel: starting velocity in unit of pulse per second
max_vel: maximum velocity in unit of pulse per second
Tacc: specified acceleration time in unit of second
Tlacc: specified linear acceleration section of s-curve in second
ratio: ratio of (feedback resolution)/(command resolution)

@ Return Code

ERR_NoError

6.11 Home Return

@ Name

_8132_set_home_config – Set the configuration for home return.
_8132_home_move – Perform a home return move.

@ Description

_8132_set_home_config:

Configure the logic of origin switch and index signal needed for home_move() function. If you need to stop the axis after EZ signal is active(home_mode=1 or 2), you should keep placing ORG signal in the ON status until the axis stop. If the pulse width of ORG signal is too short to keep it at ON status till EZ goes ON, you should select the org_latch as enable. The latched condition is cancelled by the next start or by disabling the org_latch. Three home return modes are available. Refer to Chapter4.1.5 for the setting of home_mode control.

_8132_home_move:

This function will cause the axis to perform a home return move according to the setting of **set_home_config()** function. The direction of moving is determined by the sign of velocity parameter(svel, mvel). Since the stopping condition of this function is determined by *home_mode* setting, user should take care to select the initial moving direction. Or user should take care to handle the condition when limit switch is touched or other conditions that is possible causing the axis to stop. Executing v**_stop()** function during **home_move()** can also cause the axis to stop.

*@ **Syntax***

C/C++ (DOS, Windows 95/NT)

U16 _8132_set_home_config(I16 axis, I16 home_mode, I16 org_logic, I16 org_latch, I16 EZ_logic)

U16 _8132_home_move(I16 axis, F64 svel, F64 mvel, F64 accel)

Visual Basic (Windows 95/NT)

B_8132_set_home_config (ByVal axis As Long, ByVal home_mode As Long, ByVal org_logic As Long, ByVal org_latch As Long, ByVal EZ_logic As Long) As Integer

B_8132_home_move (ByVal axis As Long, ByVal str_vel As Double, ByVal max_vel As Double, ByVal accel As Double) As Integer

*@ **Argument***

axis: axis number designated to configure and perform home returning

home_mode: stopping modes for home return.
home_mode=0, ORG active only.
home_mode=1, ORG active and then EZ active to stop, high speed all the way.
home_mode=2, ORG active and then EZ active to stop, high speed till ORG active then low speed till EZ active.

org_logic: Action logic configuration for ORG signal
org_logic=0, active low; org_logic=1, active high

org_latch: Latch state control for ORG signal
org_latch=0, don't latch input; org_latch=1, latch input.

EZ_logic: Action logic configuration for EZ signal
EZ_logic=0, active low; EZ_logic=1, active high.

*@ **Return Code***

ERR_NoError

6.12 Manual Pulser Motion

*@ **Name***

_8132_set_manu_iptmode – Set pulser input mode and operation mode

_8132_manu_move – Begin a manual pulser movement

_8132_set_manu_axis – Select manual pulser axis

_8132_set_step_unit – Set manual pulser ratio

*@ **Description***

_8132_set_manu_iptmode:

Four types of pulse input modes can be available for pulser or hand wheel. User can also move two axes simultaneously with one pulser by selecting the operation mode to ***common mode***. Or move the axes independently by selecting the operation mode to ***independent mode***.

_8132_manu_move:

Begin to move the axis according to manual pulser input as this command is written. The maximum moving velocity is limited by **mvel** parameter. Not until the ***v_stop()*** command is written won't system end the manual move mode.

_8132_set_manu_axis:

Choose the control axis for manual pulser. User can set which axis will move by manual pulser or stop the manual pulser output.

_8132_set_step_unit:

Set the unit number of output pulses to one manual input pulse.

@ *Syntax*

C/C++ (DOS, Windows 95/NT)

```
U16 _8132_set_manu_iptmode(I16 axis, I16 ipt_mode, I16 op_mode)
U16 _8132_manu_move(I16 axis, F64 mvel)
U16 _8132_set_manu_axis(I16 cardno, I16 manu_axis )
U16 _8132_set_step_unit(I16 axisno, I16 unit)
```

Visual Basic (Windows 95/NT)

```
B_8132_set_manu_iptmode (ByVal axis As Long, ByVal manu_iptmode As
        Long, ByVal op_mode As Long) As Integer
B_8132_manu_move (ByVal axis As Long, ByVal max_vel As Double) As
        Integer
B_8132_set_manu_axis (ByVal cardno as integer , byVal manu_axis as
        integer ) As Integer
B_8132_set_step_unit(ByVal axisno as integer, ByVal unit as integer) As
        Integer
```

@ *Argument*

axis: axis number designated to start manual move

ipt_mode: setting of manual pulser input mode from PA and PB pins
- ipt_mode=0, 1X AB phase type pulse input.
- ipt_mode=1, 2X AB phase type pulse input.
- ipt_mode=2, 4X AB phase type pulse input.
- ipt_mode=3, CW/CCW type pulse input.

op_mode: common or independent mode selection
- op_mode=0, Independent for each axis
- op_mode=1,PAX, PBX common for PAY, PBY or PAZ, PBZ common for PAU, PBU.

mvel: limitation for maximum velocity

manu_axis: select manual pulser output axis:
- manu_axis=0, no axis output from manual pulser
- manu_axis=1, axis0 as manual pulser output
- manu_axis=2, axis1 as manual pulser output
- manu_Axis=3, both axis0 and axis1 as manual pulser output

unit: pulser ratio

@ Example

```
_8132_set_manu_iptmode(0,2,0); // set 4X AB Phase signal
_8132_set_manu_Axis(0,0); // user axis 0 as output
_8132_manu_move(0,10000); // active pulser
.
.
.
_8132_v_stop(0,0.1); // stop pulser move
```

@ Return Code

ERR_NoError

6.13 Motion Status

@ Name

_8132_motion_done – Return the status when a motion is done

@ Description

_8132_motion_done:

Return the motion status of PCI-8132. position.

Definition of return value is as following:

Return value =

0 : the axis is busying.

1: a movement is finished

2: the axis stops at positive limit switch

3: the axis stops at negative limit switch

4: the axis stops at origin switch

5: the axis stops because the ALARM signal is active

The following code demonstrates how to utilize this function:

```
_8132_start_a_move(axis_x, pos1, svel, mvel, Tacc);
// Begin a trapezoidal velocity   profile motion
while(!motion_done(axis_x)) // Wait for completion of
        {
                        // start_a_move()
        if(kbhit())
        {
                // Keyboard hit to escape the
    getch();
                // WHILE loop
            exit(1);
        }
}
```

@ Syntax

C/C++ (DOS, Windows 95/NT)

U16 _8132_motion_done(I16 axis)

Visual Basic (Windows 95/NT)

B_8132_motion_done (ByVal axis As Integer) As Integer

@ Argument

axis: axis number of motion status

@ Return Code

ERR_NoError

6.14 Servo Drive Interface

@ Name

_8132_set_alm_logic – Set alarm logic and alarm mode
_8132_set_inp_logic – Set In-Position logic and enable/disable
_8132_set_sd_logic – Set slow down point logic and enable/disable
_8132_set_erc_enable – Set ERC pin output enable/disable
_8132_set_sd_stop_mode – Set slow down mode

@ Description

_8132_set_alm_logic:

Set the active logic of **ALARM** signal input from servo driver. Two reacting modes are available when **ALARM** signal is active.

_8132_set_inp_logic:

Set the active logic of **In-Position** signal input from servo driver. Users can select whether they want to enable this function. Default state is disabled.

_8132_set_sd_logic:

Set the active logic and latch control of **SD** signal input from mechanical system. Users can select whether they want to enable this function. Default state is disabled.

_8132_set_erc_enable:

You can set ERC pin output enable/disable by this function. Default state is enabled.

_8132_set_sd_stop_mode:

There are two types in slow down action. One is slow down to starting velocity. The other is slow down to stop.

@ Syntax

C/C++ (DOS, Windows 95/NT)

```
U16 _8132_set_alm_logic(I16 axis, I16 alm_logic, I16 alm_mode)
U16 _8132_set_inp_logic(I16 axis, I16 inp_logic, I16 inp_enable)
U16 _8132_set_sd_logic(I16 axis, I16 sd_logic, I16 sd_latch, I16 sd_enable)
U16 _8132_set_erc_enable(I16 axis, I16 erc_enable)
U16 _8132_set_sd_stop_mode(I16 axis, I16 sd_mode)
```

Visual Basic (Windows 95/NT)

```
B_8132_set_alm_logic  (ByVal axis As Long, ByVal alm_logic As Long,
        ByVal alm_mode As Long) As Integer
```

B_8132_set_inp_logic (ByVal axis As Long, ByVal inp_logic As Long, ByVal inp_enable As Long) As Integer
B_8132_set_sd_logic (ByVal axis As Long, ByVal sd_logic As Long, , ByVal sd_latch As Long, ByVal sd_enable As Long) As Integer
B_8132_set_erc_enable(ByVal axis As Integer, ByVal erc_enable As Long) As Integer
B_8132_set_sd_stop_mode(ByVal axis As Integer, ByVal sd_mode As Integer) As Integer

@ *Argument*

axis: axis number designated to configure
alm_logic: setting of active logic for ALARM signal
alm_logic=0, active LOW.
alm_logic=1, active HIGH.
inp_logic: setting of active logic for INP signal
inp_logic=0, active LOW.
inp_logic=1, active HIGH.
sd_logic: setting of active logic for SD signal
sd_logic=0, active LOW.
sd_logic=1, active HIGH.
sd_latch: setting of latch control for SD signal
sd_logic=0, do not latch.
sd_logic=1, latch.
alm_mode: reacting modes when receiving ALARM signal.
alm_mode=0, motor immediately stops.
alm_mode=1, motor decelerates then stops.
inp_enable: INP function enable/disable
inp_enable=0, Disabled
inp_enable=1, Enabled
sd_enable: Slow down point function enable/disable
sd_enable=0, Disabled
sd_enable=1, Enabled
erc_enable: ERC pin output enable/disable
erc_enable=0, Disabled
erc_enable=1, Enabled
sd_mode: sd_move=0, slow down to starting velocity
sd_mode=1, slow down to stop

@ *Return Code*

ERR_NoError

6.15 I/O Control and Monitoring

@ Name

_8132_Set_SVON – Set state of general purpose output pin
_8132_get_io_status – Get all the I/O status of PCI-8132

@ Description

_8132_Set_SVON:

Set the High/Low output state of general purpose output pin **SVON**.

_8132_get_io_status:

Get all the I/O status for each axis. The definition for each bit is as following:

Bit	Name	Description
0	+EL	Positive Limit Switch
1	-EL	Negative Limit Switch
2	+SD	Positive Slow Down Point
3	-SD	Negative Slow Down Point
4	ORG	Origin Switch
5	EZ	Index signal
6	ALM	Alarm Signal
7	SVON	SVON of PCL5023 pin output
8	RDY	RDY pin input
9	INT	Interrupt status
10	ERC	ERC pin output
11	INP	In-Position signal input

@ Syntax

C/C++ (DOS)

```
U16 _8132_Set_SVON(I16 axis, I16 on_off)
U16 _8132_get_io_status(I16 axis, U16 *io_status)
```

C/C++ (Windows 95/NT)

```
U16 _8132_Set_SVON(I16 axis, I16 on_off)
U16 _8132_get_io_status(I16 axis, U16 *io_status)
```

Visual Basic (Windows 95/NT)

```
B_8132_Set_SVON    (ByVal axis As Long, ByVal on_off As Long) As
        Integer
B_8132_get_io_status    (ByVal axis As Integer, io_sts As Integer) As
        Integer
```

@ Argument

axis: axis number for I/O control and monitoring
on_off: setting for SVON pin digital output
on_off=0, SVON is LOW.

on_off=1, SVON is HIGH.
***io_status**: I/O status word. Where "1' is ON and "0" is OFF. ON/OFF state is read based on the corresponding set logic.

@ Return Code

ERR_NoError

6.16 Position Control

@ Name

_8132_set_position – Set the actual position.
_8132_get_position – Get the actual position.
_8132_set_command – Set the current command position.
_8132_get_position – Get the current command position.

@ Description

_8132_set_position()
changes the current actual position to the specified position.

_8132_get_position()
reads the current actual position. Note that when feedback signals is not available in the system, thus external encoder feedback is *Disabled* in **set_cnt_src()** function, the value gotten from this function is command position.

_8132_set_command()
changes the command position to the specified command position.

_8132_get_command()
reads the current command position.

@ Syntax

C/C++ (DOS, Windows 95/NT)

```
U16 _8132_set_position(I16 axis, F64 pos)
U16 _8132_get_position(I16 axis, F64 *pos)
U16 _8132_set_command(I16 axis, F64 pos)
U16 _8132_get_command(I16 axis, F64 *pos)
```

Visual Basic (Windows 95/NT)

```
B_8132_get_position   (ByVal axis As Integer, pos As Double) As Integer
B_8132_set_position   (ByVal axis As Integer, ByVal pos As Double) As
        Integer
B_8132_get_command    (ByVal axis As Integer, pos As Double) As Integer
B_8132_set_command    (ByVal axis As Integer, ByVal pos As Double) As
        Integer
```

@ Argument

axis: axis number designated to set and get position.
pos: actual position or command position

@ Return Code

ERR_NoError

6.17 Interrupt Control

@ Name

_8132_Set_INT_ENABLE – Set interrupt enable
_8132_INT_Enable – Set interrupt enable
_8132_INT_Disable – Set interrupt disable
_8132_Set_INT_Control – Set interrupt event handle
_8132_set_int_factor – Set interrupt generating factors
_8132_get_int_axis – Get the axis which generates interrupt
_8132_get_int_status – Get the interrupting status of axis

@ Description

_8132_Set_INT_ENABLE:

This function is used to enable interrupt generating to host PC. .(This function just support DOS only.)

_8132_INT_Enable:

This function is used to enable interrupt generating to host PC.(This function just support Window 95 and Window NT only.)

_8132_INT_Disable:

This function is used to disable interrupt generating to host PC.(This function just support Window 95 and Window NT only.)

_8132_Set_INT_Control :

This function is used to assign the window INT event.(This function just support Window 95 and Window NT only.)

_8132_set_int_factor:

This function allows users to select factors to initiate the INT signal. PCI-8132 can generate INT signal to host PC by setting the relative bit as 1. The definition for each bit is as following:

Bit	Interrupt Factor
0	Stop with the EL signal
1	Stop with the SD signal
2	Stop with the ALM signal
3	Stop with the STP signal
4	Should be set to 0
5	Completion of home return
6	Completion of preset movement
7	Completion of interpolating motion for two axes: (X & Y) or (Z & U)
8~12	X(should be set to 0)
13	when ***v_stop()*** function stop the axis

14	EA/EB, PA/PB encoder input error
15	start with STA signal
16	Completion of acceleration
17	Start of deceleration
18~22	Should be Set to 0
23	RDY active (AP3 of PCL5023 change from 1 to 0)
24~31	Should be set to 0

Note: Bit 14: The interrupt is generated when pins EA and EB, or PA and PB change simultaneously. It means there has an encoder input error.

get_int_axis:

This function allows user to identify which axis generates the INT signal to host PC. (**This function is for DOS only**)

get_int_status:

This function allows user to identify what kinds of interrupt is generated.

After user gets this value, the status register will be cleared to 0. The return value is a 32 bits unsigned integer and the definition for each bit is as following:

Bit	Interrupt Type
0	Stop with the +EL signal
1	Stop with the –EL signal
2	Stop with the +SD signal
3	Stop with the –SD signal
4	Stop with the ALM signal
5	Stop with the STP signal
6	Comparator Active
7	Always 0
8	Stop with v_stop() command
9	Stop with home return completed
10	Always 0
11	Stop with preset movement completed
12	Stop with EA/EB input error
13	Always 0
14	Stop with PA/PB input error
15	Start with STA signal
16	Acceleration Completed
17	Deceleration Started
18~22	Always 0
23	RDY active(AP3 of PCL5023 change from 1 to 0)
24~31	Always 0

@ *Syntax*

C/C++ (DOS)

U16 _8132_Set_INT_ENABLE(U16 cardNo, U16 intFlag)
U16 _8132_set_int_factor(U16 axis, U32 int_factor)
U16 _8132_get_int_axis(U16 *int_axis)
U16 _8132_get_int_status(U16 axis, U32 *int_status)

C/C++ (Windows 95/NT)

U16 _8132_INT_Enable (I16 cardNo, HANDLE *phEvent)
U16 _8132_INT_Disable (I16 cardNo)
U16 _8132_Set_INT_Control(U16 cardNo, U16 int1Flag, U16 int2Flag)
U16 _8132_set_int_factor(U16 axis, U32 int_factor)
U16 _8132_get_int_status(I16 axis, U32 *int_status)

Visual Basic (Windows 95/NT)

_8132_INT_Enable (ByVal cardNo As Long, phEvent As Long)
_8132_INT_Disable (ByVal cardNo As Long) As Integer
_8132_Set_INT_Control (ByVal cardno As Integer, ByVal int1Flag As Integer, ByVal int2Flag As Integer)
_8132_set_int_factor (ByVal axis As Integer, ByVal int_factor As Long) As Integer
_8132_get_int_status (ByVal axis As Long, int_status As Long) As Integer

@ *Argument*

cardNo: card number 0,1,2,3…
axis: axis number 0,1,2,3,4…
intFlag: int flag, 0 or 1
phEvent: event or event array for interrupt axis (Windows)
int_factor: interrupt factor, refer to previous interrupt factor table
int_axis: interrupt axis number (the return value)
int_status: interrupt factor (the return value), refer to previous interrupt type table
int1Flag: INT flag for interrupt mentioned in manual, 0: disable or 1: enable
int2Flag: INT flag for compare interrupt, 0: disable or 1: enable

@ *Return Code*

ERR_NoError

6.18 Digital Input/Output Control

@ Name

_8132_DO – Set output value

_8132_DI – Get input value

@ Description

_8132_DO:

Set a 16-bits value to PCI-8132's digital output channels. Each bit of this value represents a high/low value for one channel.

_8132_DI:

Get a 16-bits value from PCI-8132's digital input channels. Each bit of this value represents a high/low value for one channel.

@ Syntax

C/C++ (DOS)

U16 _8132_DO(U16 axis, U16 DoData)
U16 _8132_DI(U16 axis, U16 *DiData)

C/C++ (Windows 95/NT)

U16 _8132_DO(U16 axis, U16 DoData)
U16 _8132_DI(U16 axis, U16 *DiData)

Visual Basic (Windows 95/NT)

B_8132_DO(ByVal axis As Integer, ByVal DoData As Long) As Integer
B_8132_DI(ByVal axis As Long, DiData As Long) As Integer

@ Argumen

axis: axis number 0,1,2,3,4…
DoData: a 16-bits output value
DiData: a 16-bits input value

@ Return Code

ERR_NoError

6.19 Position Compare Control

@ Name

_8132_Get_CompCnt – Get counter value from comparator
_8132_Set_CompCnt – Set counter value in comparator
_8132_Set_CompMode – Set compare mode
_8132_Set_CompData – Set comparator value
_8132_Get_CompData – Get current comparator value
_8132_Set_CompInt – Enable comparator Interrupt
_8132_Set_CompHome – Set comparator origin
_8132_Get_CompSts – Get comparator status
_8132_Build_Comp_Table – Build compare table
_8132_Set_Comp_Table – Enable/Disable compare table
_8132_Build_Comp_Function – Build a linear compare table by a function

@ Description

_8132_Get_CompCnt /_8132_Set_CompCnt:

Read or write the counter value in FPGA comparator on PCI-8132.

_8132_Set_CompData / _8132_Get_CompData:

Read or write the current value for position compare

_8132_Set_CompMode:

Set position compare rule for one axis. User can choose the compare direction from this function

_8132_Set_CompInt:

Enable/disable the comparator interrupt. If user uses a compare table for “on the fly compare”, the comparator interrupt must be enabled. Interrupt will trigger kernel driver to load next compare point and send a Windows Event to notify user’s AP. If the frequency of comparator output is too high, the Windows Event won’t be received by AP without lost but the hardware trigger will be send correctly without delay.

_8132_Set_CompHome:

Reset the comparator’s counter to zero. This function usually follows by home_move() to make sure that two counter are the same before any motion.

_8132_Get_CompSts:

Get current status of comparator

_8132_Build_Comp_Table:

PCI-8132 provides a convenient interface for user to input their compare points. User can pass an array pointer to this function to notify PCI-8132. The maximum points of this table are 1024 long integer value.

_8132_Set_Comp_Table:

Once user builds a compare table by *_8132*_Build_Comp_Table(), he

can use this function to control the table active or not.

_8132_Set_Comp_Function:

This is an alternative way to set up compare data if user's compare points are equal interval. It is no size limit if user uses this method.

@ *Syntax*

C/C++ (DOS)

```
U16 _8132_Get_CompCnt(U16 axis, double *act_pos);
U16 _8132_Set_CompCnt(U16 axis, double cnt_value);
U16 _8132_Set_CompMode(U16 axis, I16 comp_mode);
U16 _8132_Set_CompData(U16 axis, double comp_data);
U16 _8132_Get_CompData(U16 axis, double *comp_data);
U16 _8132_Set_CompInt(U16 axis, U16 enable);
U16 _8132_Set_CompHome(U16 axis);
U16 _8132_Get_CompSts(U16 cardNo, U16 *Comp_Sts);
U16 _8132_Build_Comp_Table(U16 axis, I32 *table, I16 Size);
U16 _8132_Set_Comp_Table(U16 axis, U16 Control);
U16 _8132_Build_Comp_Function(U16 axis,I32 Start,I32 End,I32 Interval);
```

C/C++ (Windows 95/NT)

```
U16   PASCAL   _8132_Get_CompCnt(U16 axis, double *act_pos);
U16   PASCAL   _8132_Set_CompCnt(U16 axis, double cnt_value);
U16   PASCAL   _8132_Set_CompMode(U16 axis, I16 comp_mode);
U16   PASCAL   _8132_Set_CompData(U16 axis, double comp_data);
U16   PASCAL   _8132_Get_CompData(U16 axis, double *comp_data);
U16   PASCAL   _8132_Set_CompInt(U16 axis, U16 enable);
U16   PASCAL   _8132_Set_CompHome(U16 axis);
U16   PASCAL   _8132_Get_CompSts(U16 cardNo, U16 *Comp_Sts);
U16   PASCAL   _8132_Build_Comp_Table(U16 axis, I32 *table, I16 Size);
U16   PASCAL   _8132_Set_Comp_Table(U16 axis, U16 Control);
U16   PASCAL   _8132_Build_Comp_Function(U16 axis,I32 Start,I32
          End,I32 Interval);
```

Visual Basic (Windows 95/NT)

```
B_8132_Set_CompInt(ByVal axis As Integer, ByVal enable As Integer) As
      Integer
B_8132_Get_CompData ByVal axis As Integer, comp_data As Double) As
      Integer
B_8132_Set_CompData (ByVal axis As Integer, ByVal comp_data As
      Double) As Integer
B_8132_Set_CompMode (ByVal axis As Integer, ByVal comp_mode As
      Integer) As Integer
B_8132_Set_CompCnt (ByVal axis As Integer, ByVal cnt_value As Double)
      As Integer
B_8132_Get_CompCnt (ByVal axis As Integer, act_pos As Double) As
      Integer
B_8132_Set_CompHome (ByVal axis As Integer) As Integer
B_8132_Build_Comp_Table (ByVal axis As Integer, table As Long, ByVal
```

Size As Integer) As Integer
B_8132_Set_Comp_Table (ByVal axis As Integer, ByVal Control As Integer) As Integer
B_8132_Build_Comp_Table(ByVal axis As Integer, ByVal Start As Long, ByVal End As Long, ByVal Interval As Long) AS Integer

@ *Argumen*

axis: axis number 0,1,2,3,4…
enable: 1 means enable, 0 means disable
comp_data: comparator value
cnt_value: counter value
comp_mode: comparator mode
0=increasing (counter > compare value)
1=equal (counter = compare value)
2=decreasing (counter < compare value)

***table:** compare table pointer
size: compare table size
control: 0 means disable
1 means compare points is from compare table
2 means compare points is from linear function
comp_sts: the definition are as follows:
bit0: CMP1 Out Status , Low=0 and high=1
bit1: CMP2 Out Status , Low=0 and high=1
bit2~bit6 not use
bit7: Interrupt happened=1, not happened=0
start: compare function start point
end: compare function end point
interval: compare function incremental size

@ *Return Code*

ERR_NoError

7

Connection Example

This chapter shows some connection examples between PCI-8132 and servo drivers and stepping drivers.

7.1 General Description of Wiring

Figure 7.1 is a general description of all the connectors of PCI-8132. Only connection of one of 2 axes is shown.

CN1: Receives +24V power from external power supply.

CN2 :Main connection between PCI-8132 and pulse input servo driver or stepping driver.

CN3: Connector for simultaneously start or stop multiple PCI-8132 cards.

Figure 7.2 shows how to integrate PCI-8132 with a physical system.

Description of PCI-8132 Indexer Pinouts

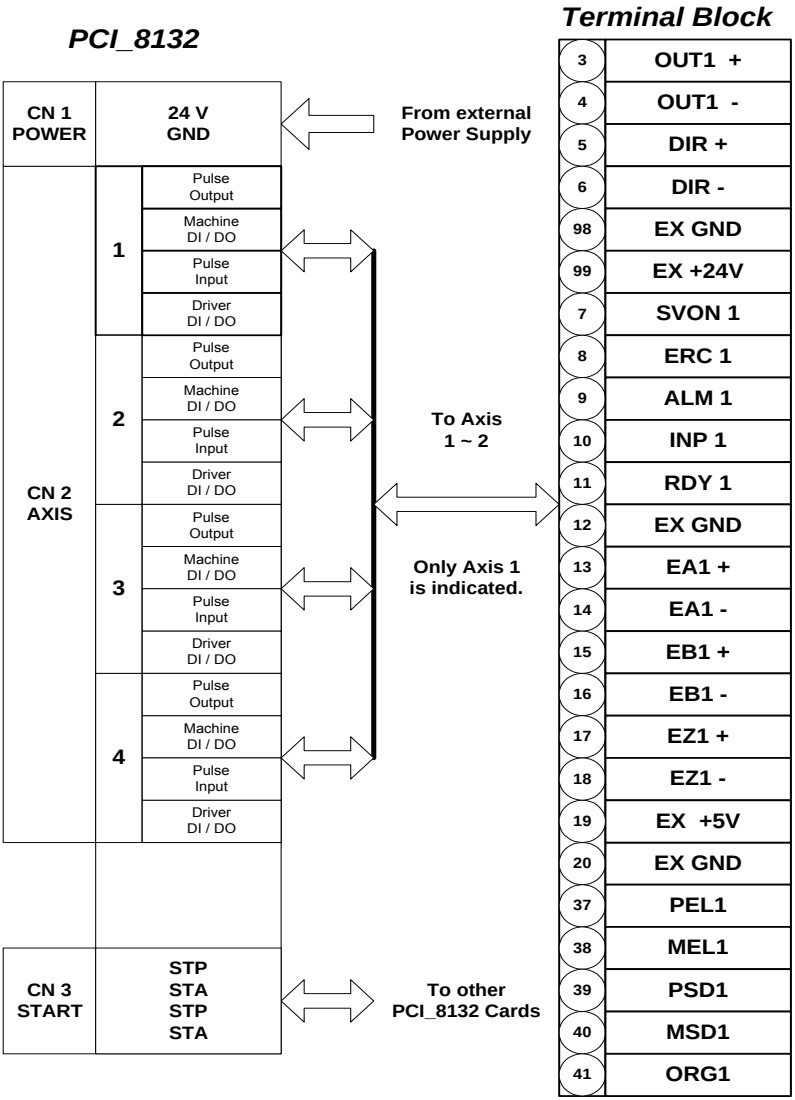


Figure 7.1 General Description of Wiring

Wiring of PCI-8132 with Servo Driver

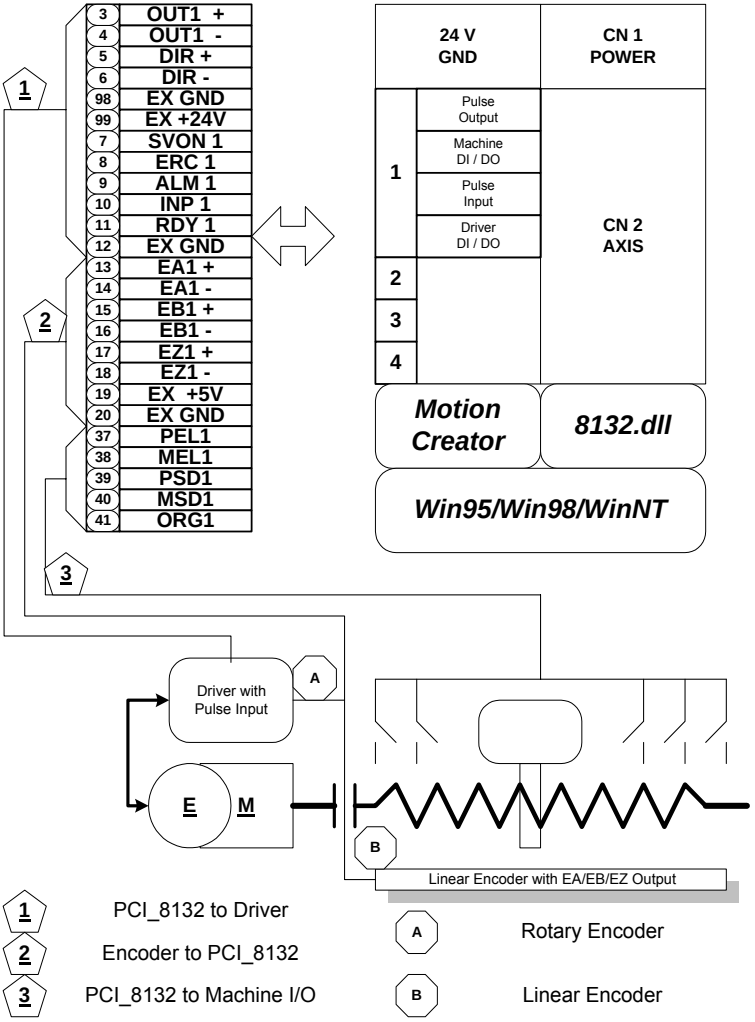


Figure 7.2 System Integration with PCI-8132

7.2 Connection Example with Servo Driver

In this section, we use ***Panasonic Servo Driver*** as an example to show how to connect it with ***PCI-8132***. Figure 7.3 show the wiring.

Note that:

1. For convenience' sake , the drawing shows connections for one axis only.
2. Default pulse output mode is **OUT/DIR** mode; default input mode is 4X **AB phase** mode. Anyway, user can set to other mode by software function.
3. Since most general purpose servomotor driver can operates in ***Torque Mode; Velocity Mode; Position mode.*** For linking with PCI-8132, user should set the operating mode to Position Mode. By setting servo driver to this mode, user can use PCI-8132 to perform either ***Position Control*** or ***Velocity Control***.
4. The ***Deviation Counter Clear*** input for Panasonic Driver is line drive type where ***ERC*** output of PCI-8132 is open collector type. So a little circuit is required for interfacing.

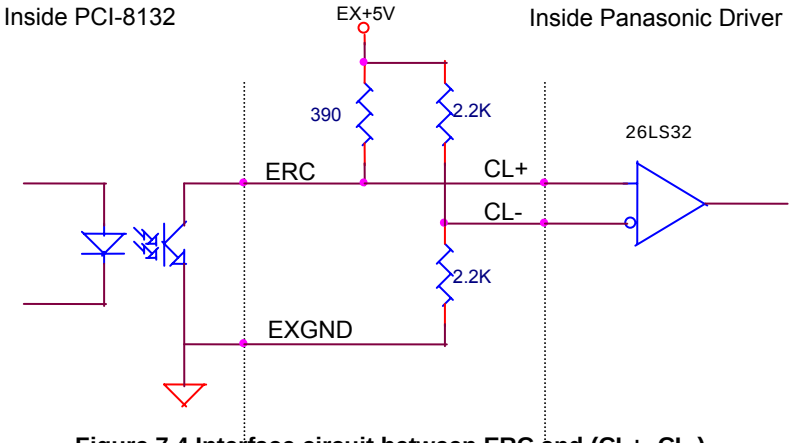


Figure 7.4 Interface circuit between ERC and (CL+, CL-)

Wiring of PCI-8132 with Panasonic MSD

PCI_8132 Axis 1 **Servo Driver**

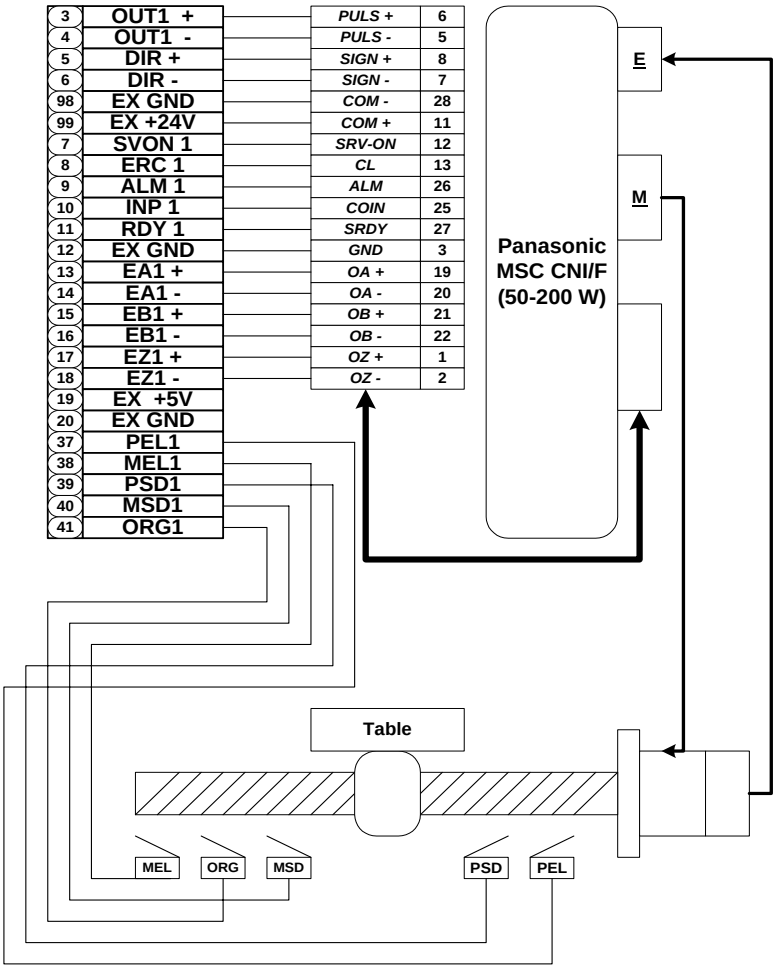


Figure 7.3 Connection of PCI-8132 with Panasonic Driver

Warranty Policy

Thank you for choosing ADLINK. To understand your rights and enjoy all the after-sales services we offer, please read the following carefully:

1. Before using ADLINK's products please read the user manual and follow the instructions exactly.
2. When sending in damaged products for repair, please attach an RMA application form.
3. All ADLINK products come with a two-year guarantee, repaired free of charge.

- The warranty period starts from the product's shipment date from ADLINK's factory.
- Peripherals and third-party products not manufactured by ADLINK will be covered by the original manufacturers' warranty.
- End users requiring maintenance services should contact their local dealers. Local warranty conditions will depend on local dealers.

4. This warranty will not cover repair costs due to:
   a. Damage caused by not following instructions.
   b. Damage caused by carelessness on the users' part during product transportation.
   c. Damage caused by fire, earthquakes, floods, lightening, pollution, other acts of God, and/or incorrect usage of voltage transformers.
   d. Damage caused by unsuitable storage environments (i.e. high temperatures, high humidity, or volatile chemicals.
   e. Damage caused by leakage of battery fluid.
   f. Damage from improper repair by unauthorized technicians.
   g. Products with altered and/or damaged serial numbers.
   h. Other categories not protected under our guarantees.
5. Customers are responsible for shipping costs to transport damaged products to our company or sales office.
6. To ensure the speed and quality of product repair, please download a RMA application form from our company website: www.adlinktech.com. Damaged products with attached RMA forms receive priority.

For further questions, please contact our FAE staff.

ADLINK: service@adlinktech.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP